QB PLUS
OWNER'S MANUAL

PROFESSIONAL
BASIC PROGRAMMING TOOLS

QB Plus
Desk Accessories for the QuickBASIC Programmer

Version 1.0

CRESCENT SOFTWARE, INC.
32 SEVENTY ACRES
WEST REDDING, CT 06896
(203) 438-5300
Crescent's Support BBS: (203) 426-5958

LICENSE AGREEMENT

Crescent Software, Inc. grants a license to use the enclosed software and printed documentation to the original purchaser. Copies may be made for back–up purposes only. Copies made for any other purpose are expressly prohibited, and adherence to this requirement is the sole responsibility of the purchaser. Source code and libraries for any component of the QB Plus program may not be distributed under any circumstances. This license may be transferred to a third party only if all existing copies of the software and documentation are also transferred.

WARRANTY INFORMATION

Crescent Software, Inc. warrants that this product will perform as advertised. In the event that it does not meet the terms of this warranty, and only in that event, Crescent Software, Inc. will replace the product or refund the amount paid, if notified within 30 days of purchase. Proof of purchase must be returned with the product, as well as a brief description of how it fails to meet the advertised claims.

CRESCENT SOFTWARE'S LIABILITY IS LIMITED TO THE PURCHASE PRICE. Under no circumstances shall Crescent Software or the authors of this product be liable for any incidental or consequential damages, nor for any damages in excess of the original purchase price.

TABLE OF CONTENTS

Chapter 1: Introduction

Introduction ..... 1 – 1
Description of Built-in Accessories ..... 1 – 1
Memory Viewer ..... 1 – 1
Macro Keystrokes ..... 1 – 1
Extended .EXE Builder ..... 1 – 2
External Debugger Support ..... 1 – 2
Execution Profiler ..... 1 – 2
Switch Programs (Add-ons) ..... 1 – 3
Required Hardware and Software ..... 1 – 3
Installation and Start Up ..... 1 – 4
Installation ..... 1 – 5
Start up ..... 1 – 6
QB Plus Command Line Options ..... 1 – 7

Chapter 2: Extended Executable File Builder

Extended Executable File Builder ..... 2 – 1
The Build .EXE Window ..... 2 – 1
"Starter Set" Default Options ..... 2 – 2
Edit Commands ..... 2 – 3
Format and Syntax ..... 2 – 4
Options List ..... 2 – 5
Saving the Build Options ..... 2 – 6
Building the Executable File ..... 2 – 6
QB Plus Drives, Directories and Paths ..... 2 – 7
QB Plus Tips and Traps ..... 2 – 7

Chapter 3: External Debugger

External Debugger ..... 3 – 1

Chapter 4: Macro Keystrokes

Macro Keystrokes ..... 4 – 1
Recording Keystrokes ..... 4 – 1

Chapter 4: Macro Keystrokes (Continued)

Keystroke Playback ..... 4 – 3
Viewing Recorded Keystrokes ..... 4 – 4
Saving Recorded Keystrokes ..... 4 – 5
Loading Pre-recorded Keystrokes From Disk ..... 4 – 6
Joining One Set of Keystrokes to Another ..... 4 – 7

Chapter 5: Program Execution Profiler

Program Execution Profiler ..... 5 – 1
Profiling Set Up ..... 5 – 2
Profiling Analysis ..... 5 – 3
Time Percentages ..... 5 – 4
Call Percentages ..... 5 – 4
Call Duration ..... 5 – 4
Printing the Analysis ..... 5 – 5
Optimizing ..... 5 – 5
Profiler Considerations ..... 5 – 6
Automating Profile Sampling Runs ..... 5 – 7
Profiling and Macro Keys ..... 5 – 7
Interpreting Changes ..... 5 – 7
QB Plus Profiler Overhead ..... 5 – 8
Potential Sampling Errors—Halt/Resume Sequence ..... 5 – 8
Potential Sampling Errors—System Timer Reprogramming ..... 5 – 9
Sample Rate Harmonics ..... 5 – 9
QuickBASIC History, Watches, and Breakpoints ..... 5 – 10
Cumulative Sampling Runs ..... 5 – 10
QB Environment Versus Executable Programs ..... 5 – 10
Profiler Capacity ..... 5 – 12

Chapter 6: Memory Viewer

Memory Viewer ..... 6 – 1
Caution and Limitations ..... 6 – 1
Caution: I/O Ports: ..... 6 – 1
Hardware Limitations ..... 6 – 2
Memory View Screen ..... 6 – 2
Getting Help ..... 6 – 3
Getting Around in Memory ..... 6 – 3

Chapter 6: Memory Viewer (Continued)

Movement Keys 6-4
Entering Addresses 6-4
Out of Range Addresses 6-5
Viewing Modes 6-5
Byte 6-5
Word 6-5
Integer 6-7
Long 6-7
Vector 6-8
ASCII 6-9
Ports (Caution) 6-10
CMOS 6-10
Real Time 6-10
Getting Information 6-11
Conventional Memory Information 6-11
Expanded Memory Information 6-11
Extended Memory Information 6-12
EMS/XMS Handle List 6-13
Addressing Modes 6-13
Conventional Addressing 6-13
Linear Addressing 6-14
Expanded Memory Addressing 6-15
XMS Memory Addressing 6-16

Chapter 7: The Program Switcher and Add-On Accessories

The Program Switcher and Add-On Accessories 7-1
Preserving QB Plus/QB Screen Image 7-3

Chapter 8: Change Settings

Change Settings 8-1
Debugger Settings 8-1
Macro Key Customization 8-2
Other Settings 8-3
File Swap Name 8-3
On/Off Toggle Switch Box 8-3

Chapter 9: Ending The Accessories

Ending the Accessories 9-1

APPENDICES

QB Plus Error Condition Codes A-1
Macro Key Technical Information A-2
Messages A-3
Problem Conditions A-7

Introduction

QB Plus is a collection of software accessories developed especially for the QuickBASIC programmer. QB Plus extends the QuickBASIC programming environment by loading with it and remaining two keystrokes away while editing the QuickBASIC program in memory.

Provided with QB Plus are accessories to view the contents of memory; to record and playback keystrokes; to create executable programs using any of the compiler and linker options; debug them using an external debugger; and profile the execution of QuickBASIC programs to identify bottlenecks. Besides these built-in tools, the BASIC programmer can readily access other programs from within QB Plus.

QB Plus is designed to work with QuickBASIC version 4.5; and the Basic Professional Development System versions 7.0 and 7.1.

Description of Built-in Accessories

Memory Viewer

The QB Plus memory viewer opens a window into the PC to display the contents of conventional memory, I/O Port registers, the CMOS configuration area, and if present, expanded and extended memory. Memory values may be displayed as hex bytes, 16-bit hex words, 32-bit hex long integers, signed decimal integers, ASCII characters, and segment:offset style hex addresses. A real time mode provides a continually updated display of changing memory values.

In addition, an information window provides conventional memory locations of interest, as well as information on expanded (EMS) and extended (XMS) memory drivers, EMS and XMS handles and memory allocations.

Macro Keystrokes

QB Plus can record and playback a series of keystrokes from a single key. This lets you assign frequently used key combinations such as DECLARE FUNCTION, and then easily replay them.

Up to 40 keystrokes may be recorded and associated with any single key; or up to 36 keys can be linked together to provide up to a series of 1440 keystrokes that may be played back from a single key. Keystroke recording may be invoked only from within the QuickBASIC editor, so that it does not interfere with a basic program running in the environment. Keystroke playback, however, is fed not only to QuickBASIC to help automate repetitive program editing tasks, but may also be used to playback input sequences to a BASIC program running in the environment. Recorded keystrokes may be saved and loaded from disk.

Extended .EXE Builder

QB Plus lets you specify BC compiler and linker options that are not normally available in the QuickBASIC programming environment. This permits the creation of smaller, faster stand alone programs that take advantage of special libraries, stub files and packing and optimizing features, without having to leave the environment. QB Plus will create a "starter set" of BC and Link commands tailored to the BASIC program in memory, let you edit and add to them, and then compile and link from within QuickBASIC with a single keystroke. It also supports incremental compiling and linking to eliminate unneeded processing of program modules that have not changed since the last compile and link cycle.

The compile/link setup for a given program is preserved on disk for reuse each time that particular program is loaded into the QB Plus-enhanced environment.

External Debugger Support

QB Plus permits BASIC programmers who have an external debugger to access it from within the QuickBASIC environment, to extend the debugging support provided. With a few keystrokes you can go from editing in the environment, to building a custom executable program to testing the new executable in CodeView, and back to the editor for further changes.

Execution Profiler

When enabled, QB Plus's profiler periodically samples a QuickBASIC program running in the environment to see which SUB or FUNCTION is executing at that instant. QB Plus tabulates these samples after the program ends, showing how much of its time the program spent in each SUB and FUNCTION, the number of times each was called, and the

average time each needed to execute. With this information you can concentrate on those routines that consumed the most time, and know which optimizations will have the most effect.

Switch Programs (Add-ons)

With QB Plus, you can invoke other applications from within QuickBASIC, and thus add to QB Plus's built-in collection of accessories. QB Plus swaps QuickBASIC and the loaded BASIC source files and Quick Libraries to disk or extended or expanded memory, which frees most of conventional memory to run a specified application program. With QB Plus you can therefore run large programs that would be too big to run from QuickBASIC's DOS shell.

The TIMERUN.EXE program provided with QB Plus is an example of an external add-on program designed to complement QB Plus, and can be invoked through QB Plus's Switch Program feature.

Required Hardware and Software

QB Plus will run on most machines that QuickBASIC does, although Hercules display adapters, 3270 PC/AT displays and non-conventional memory in certain computers may not be fully supported. DOS 3.0 or greater and a hard disk are strongly recommended. Extended or expanded memory is recommended but not required.

QB Plus uses about 45K of conventional memory while QuickBASIC is running. QB Plus may therefore prevent loading or execution of BASIC programs that require a large amount of memory. Depending on options selected by the programmer, QB Plus may also use up to 80K of expanded memory for its overlays. As much as 560K of EMS or extended memory may also be needed for storing QB.EXE during execution of external applications.

QB Plus is compatible with most networks and operating system environments that permit an application to execute a child process.

There may be conflicts with system software and device drivers that operate in protected mode, dynamically relocate blocks of conventional memory, intercept and reprogram the PC's system timer, or intercept the keyboard interrupt routines and/or relocate the keyboard input buffer.

QB Plus will run in the DOS window of OS/2 1.1 and Windows 3.0; however, viewing memory above one megabyte may be blocked from QB Plus. Also, QB Plus's use of interrupt 70h and the on-board real-time clock in the Profiler may be disabled.

Installation and Start Up

Listing of files supplied with QB Plus:

QBP.EXE	The main QB Plus executable file.
QBPBUILD.EXE	The external add-on program, called by the QB Plus Build .EXE editor to compile and link a QuickBASIC program based on the contents of a .MQK file.
TIMERUN.EXE	The external add-on program that times the execution duration of another program.
ALLFREE.KQF	A macro key stroke file with all key strokes deleted.
ALLJOIN.KQF	A macro key stroke file with all key strokes deleted, but the keys consecutively joined to facilitate recording of long macros.
QBMAC.KQF	The default macro key file.
TESTPROF.KQF	Sample macro key file for MVBAS.BAS execution profiling session.
MVBAS.BAS	Source code for a BASIC version of a stand alone memory viewer.
MVIBAS.BAS	BASIC subroutine support module for MVBAS.BAS containing conventional, expanded, and extended memory information routines.
MVPROFI.BAS	BASIC subroutine support module for MVBAS.BAS containing instructions for a demonstration execution profiling session.

PASSUBS.BAS	BASIC subroutine support module for MVBAS.BAS.
TIMERUN.BAS	BASIC source code for TIMERUN.EXE.

The remaining files hold the Turbo Pascal source code for QB Plus.

Installation

QB Plus contains more than 50 different files occupying nearly 700K of disk space. All of the files are compressed in the QBPLUS.ZIP and SOURCE.ZIP files on the accompanying disk. To help you copy these files correctly onto your hard disk we have included an automated installation utility.

Installing QB Plus is very easy. Simply log on to Drive A, place the disk into that drive, and enter INSTALL at the DOS prompt.

On-screen instructions explain how to use INSTALL. It is not necessary to install the Pascal source code to use QB Plus, and we include it solely for those people who are interested. Therefore, you can simply unmark the SOURCE.ZIP file before pressing F3 to begin installation.

Note that F2 lets you see the file names inside each .ZIP file, and selectively mark or unmark them for installation. This feature lets you install only certain files if you prefer.

By default, installation is to C:\QB, though you can change that to reflect any valid drive and directory. If the directory you specify does not exist, INSTALL will create it. We recommend that you install QB Plus into the same directory in which QuickBASIC resides. If you also install the source code, those files should go into a separate directory. This avoids cluttering up your QuickBASIC directory.

If you are familiar with the PKUNZIP program, you can optionally run it manually. Entering PKUNZIP with no arguments displays a help screen that shows all of the option switches it recognizes.

Alternatively, you may place QBP.EXE and QBP.CFG and the other QB Plus support files in a drive and subdirectory separate from QB/QBX. When QB Plus is started, it will look first for QB.EXE and then QBX.EXE in the current directory; if not found it will then search

the directories listed in the DOS path, and finally search the drive and subdirectory from which QB Plus was run. You can also specify a location for QB/QBX as part of the QB Plus command line.

While it is running, QB Plus will follow a similar search process for its support modules, such as QBPBUILD.EXE. That is, first the current directory is searched, then the DOS path, then the directory where QB/QBX is located, and finally QB Plus' own location. QB Plus .KQF keystroke macro files; however, default to the current directory, unless you specify a path as part of the file name.

Separate locations for QB Plus and QB/QBX may be useful and sensible on a network to allow individual QB Plus customization.

PKUNZIP is provided under license from PKWARE, Inc.

Start up

To operate QuickBASIC with QB Plus, simply substitute QBP for QB or QBX command you now use to start QuickBASIC.

QB Plus will recognize all of the QB or QBX command line arguments. For example, if you start QB like this:

```
C>QB /H/L/AH
```

then start QB Plus as follows:

```
C>QBP /H/L/AH
```

QB Plus will first load itself, then call QuickBASIC with the command line arguments given. To show explicitly where QB.EXE or QBX.EXE is located use the QB Plus /Q: switch:

```
C>QBP /Q:\QBDIR\
```

Once QuickBASIC has started, the QB Plus pop up window will be available.

Whenever QuickBASIC is in the program editing mode waiting for input of text, you can access QB Plus by pressing its hot key combination: Shift-Control by default. QB Plus will not respond at inappropriate times, such as when QuickBASIC is expecting a command key, while it is awaiting input into one of its dialog windows, or while it is executing or compiling a program.

QB Plus Command Line Options

In addition to command line options for QuickBASIC, QB Plus will also accept its own command line arguments. It will not pass these along to QuickBASIC. The arguments may be in any order, or even intermixed with QuickBASIC arguments if you wish. All are preceded by a slash (/) character.

OPTION	MEANING
/?	QB Plus will list the command line options and then return to DOS.
/DO	When specified, QB Plus will swap the memory image of QB to disk rather than extended or expanded memory, when an external program is run in "Switch Program","Debugger" or "Build .EXE"
/NOCLS	When specified, QB Plus will leave its window on screen when an external program described above is called. Do not use this option unless you write a QB Plus add-on that displays text only within the QB Plus pop up window.
/PV	When specified, the memory viewer will read and display register data from your PC's I/O ports. Do not use this option unless you determine that such reading will not be harmful to your PC.
/NX	Prevents QB Plus entering protected mode to read extended memory directly. This is in case your PC's hardware, BIOS, or software conflict with QB Plus's method of protected mode access causing problems (system hangs, reboots, general protection faults) in the QB Plus memory viewer.
/NEO	When specified, this forces QB Plus to keep its overlays on disk and not load them into expanded memory (EMS). This frees 80K of EMS, but will slow QB Plus's operation somewhat.

/SS	Sets the QB Plus main menu pop-up key combination to Left-Shift + Right-Shift.
/KF	Loads the default macro key file, QBMAC.KQF from the current directory during start up.
/KF*keyfilename*	On startup, loads the macro key file specified in *keyfilename*. For example, the command QBP/KFTEST.KQF automatically loads the key file TEST.KQF.
/RTC	Tells QB Plus to use the AT CMOS clock for execution profile sampling. Use this switch if QB Plus fails to automatically detect the presence of a real-time clock.
/NORTC	Tells QB Plus not to use the AT CMOS clock if detected. When this switch is used profile sample rates are limited to 18 per second. Use this switch in the event of a conflict between QB Plus and another program or operating system's use of the real-time clock or interrupt 70h.
/Q:*path*\	Loads QuickBASIC from the path specified in *path*\. For example, the command QBP /Q:\QB45\ loads and executes the version of QB or QBX contained in the "\QB45" subdirectory. Note the trailing backslash.

Note that these command line options take precedence over settings stored in file by the QB Plus Change Settings Save command.

Extended Executable File Builder

Have you ever wanted to compile your QuickBASIC programs from within QuickBASIC, but needed to use different compile and link options than the menus provide? Or have you ever wanted to be able to compile only those modules of a multi-module program that have changed? Or, have you been frustrated because BASIC PDS does not let you specify stub files or a special library to help reduce the size of your programs?

With QB Plus, you can edit and test your programs in the QuickBASIC editor, yet take advantage of all the compile and link options available from the command line—without leaving the editor.

QB Plus's Build .EXE window presents the BC and LINK commands to build an executable file from the QuickBASIC source file presently in memory. You can add, change, remove options, object files, or libraries—however you like. One keystroke will then compile and link your program as you've just specified, without having to leave QuickBASIC. Another keystroke will save this setup to file for recall in your next editing session with the same program. A help screen that lists all the valid BC and LINK arguments is also available.

With this QB Plus feature, you can automatically reduce the size of your executable by using the /S compiler option, and the /F/PACKC LINK options. You can also link with stub files to eliminate unused code from the BASIC PDS runtime library. The QB Plus conditional compilation feature can also save you time by compiling only those modules which have changed since the program was last built. You can easily use libraries that substitute or add to the standard libraries supplied with QuickBASIC, such as Crescent Software's P.D.Q.

The Build .EXE Window

The Build .EXE window is primarily a line-based text editor, having a total capacity of 40 lines with 18 visible at one time.

When the Build .EXE window is first selected, QB Plus looks for a loaded BASIC source file. It then either retrieves a previously saved set of compile and link options, or constructs a starter set. If QB Plus cannot find a BASIC program in memory, you will be prompted to load one.

Up to 18 loaded QuickBASIC program modules can automatically be read from QuickBASIC's memory, and converted into a set of compile and link options for a single executable program. If your program has more than 18 modules, the extra module names will not be made a part of the starter set and you will have to enter them in the edit window manually.

QB Plus displays the BC and LINK options for the source file(s) in the Build .EXE window for you to edit as desired. The first line or lines contain the BC invocation you would type on the DOS command line to compile each component module of the program. Each line begins with "BC". Note that you may place an apostrophe (') before a BC command to disable it, or a dollar sign ($) which tells QB Plus to compile that module only if necessary. Conditional compilation and linking is discussed later in this manual.

The LINK invocation line appears below the BC lines and may also be preceded by a conditional link flag. Below that are the options to be passed to LINK: one object file per line, followed by the map file name and a library file line.

If a set of compile/link options is already present, you will be given the chance to discard them and start a new set. This should be done when you change programs if you want to use the Build .EXE window with the same program you are working with in QuickBASIC. However, this is not a requirement—once loaded, the Build .EXE options do not have to match the currently loaded program.

"Starter Set" Default Options

If options saved from a prior session are available, they are automatically loaded when the Build EXE screen is entered. If QB Plus cannot locate stored options for this program, it will construct a "starter set" of BC and LINK options to save you typing time. QB Plus uses a variety of sources:

1. **BC file module names**

   These come either from a .MAK file, or QuickBASIC's list of module names in memory. The first module name found in QuickBASIC's list is assumed to be the main module for locating any .MAK file, for naming the executable file, and for later saving of your Build options. It will appear in the first BC line.

2. **LINK .OBJ file names**

   These are taken from the module list developed above.

3. **LINK .EXE file name**

   The name of the first module found in QuickBASIC's module list is placed by QB Plus in the top BC line in the Build window, and used as the proposed executable file name in the link list.

4. **Default options for BC are /O, /S, and /T.**

   If they had been specified on the command line used to start QB Plus, /AH and /MBF, as well as any value for /C: other than 512, are added.

5. **LINK default options are /EX/SE:512/F/PACKC.**

   If PDQ.QLB is the loaded QuickLibrary, the LINK options are /NOE/NOD.

6. **The default LINK map file is NUL, which specifies that no map file is to be created.**

7. **For QuickBASIC version 4.5, the default library is BCOM45.LIB.**

   This is consistent with the default /O option for BC, specifying a stand-alone .EXE file. For QBX, it is BCL70EFR.LIB or BCL71EFR.LIB as appropriate. Any additional library is based on the loaded Quick Library originally passed on the QB Plus invocation command line.

   Note to P.D.Q. users: If the Quicklibrary is PDQ7.QLB, then BASIC7.LIB is added to the library line in QBX installations. Also, if PDQSUBS.BAS is loaded, it will also be specified for compiling and linking. You should remove the latter references as PDQSUBS.BAS is meant only for use in the environment, and not to be compiled into the final executable program.

Edit Commands

The options displayed in the Build window may be edited as you see fit. Position the cursor with the Arrows, Home and End keys.

The editor is always in Insert mode. The Enter or Insert keys will insert a full blank line on the cursor line. Del removes the character at the cursor, while Backspace-Delete removes to the left. Control+Y deletes an entire line, which goes into a Paste Buffer, available for pasting back with Shift+Insert, similar to the QB editor.

Each line is 72 characters wide (the width of the window), and there are 40 lines total. Once capacity of either has been reached, QB Plus will refuse to further insert lines or characters until some are deleted.

Esc returns you to QuickBASIC, leaving intact in memory any changes you have made. Be aware, however, that the edit buffer is shared with the Profiler sampling buffer, and using the latter will overwrite the contents of the former. Thus, be sure to save any changes you want to preserve permanently before profiling or exiting QB/QBX.

Format and Syntax

Any of the file names and options may be fully edited. The only restrictions have to do with the way QBPBUILD.EXE interprets the data for issuing compile and link commands. (Of course, since QBPBUILD.EXE is an external add-on you can customize it, or even write an entirely different version to support custom QBP compile/link formats of your own.)

1. All BC lines must appear above the LINK line, must begin with BC, 'BC, or $BC, or the drive/path and BC as in C:\QB\BC, 'C:\QB\BC, or $C:\QB\BC. Note that you must add a space after BC. The BC line must end with a semicolon. Each BC invocation must fit entirely on one line.

2. Like BC, LINK may be specified by name alone or by its full path, and the line may optionally begin with a single apostrophe or dollar sign for conditional linking. LINK must have its command line options follow it on the same line. Do not end the link line with a semi-colon.

3. Object files must appear next below the LINK line. If there is more than one object file each must have a plus sign after it, unless it is the last one in the object file list. Except as the first object file (which must be the main object module), libraries may be specified in the object file list if you wish the linker to link individual object routines from certain specified libraries.

4. Following the object files is the name of the executable file.

5. The map file specification must appear next below the executable object file name(s).

6. The last line must contain the name of the stand alone or runtime library. If more than one library is being specified, separate their names with spaces all on the same line, or each library name on an individual line followed by a plus sign similar to multiple object files. We recommend (and require with PDS 7.0 and 7.1 unless a .DEF file is being specified) that the last library name be followed with a semi-colon.

Example:

```
BC D:\QB\MAIN.BAS/X/W/O/T/C:128;
BC D:\QB\MOD1.BAS/X/W/O/T/C:128;
BC D:\QB\MOD2.BAS/X/W/O/T/C:128;
BC D:\QB\MOD3.BAS/X/W/O/T/C:128;
LINK /EX/F/PACKC
MAIN+
MOD1+
MOD2+
MOD3
D:\QB\MAIN.EXE
NUL
QB.LIB BCOM45.LIB;
```

Options List

Pressing F1 gives you not only a summary of the edit commands, but also a handy listing of valid BC and LINK command line arguments.

The versions of BC and LINK supplied with QB version 4.5 may not support all of the options listed.

Unlike QuickBASIC, QB Plus will not examine your source file for certain functions that require specific BC arguments. These include certain ERROR and event trapping statements. If your program code requires /E/X/W/V switches, you must add them yourself. Also, don't forget the other modules when you want the /E/X/W/V support incorporated. BC may not warn you of such an omission.

Likewise, ensuring that other options are appropriate and correct is up to you, such as eliminating the /O and changing BCOM45.LIB to BRUN45.LIB for the non-stand-alone .EXE version. QB Plus's purpose is to give you complete freedom with compiling and linking, so you'll have to supply the error checking yourself.

Saving the Build Options

By pressing F2, the Build setup may be saved to file. The name used for storage is the same as the main module name (the first name in the BC list), but with the extension .MQK. This file is stored in the same directory as the main module. The .MQK file is ordinary ASCII text. (If you decide to edit it with another editor, be sure to respect the 72 column width, 40 line count, or QB Plus may be unable to read it).

Like a .MAK file, you should keep the .MQK file with the source file(s). Unlike a .MAK file, however, since the .MQK file contains fully qualified file names, you will have to update it to the new source file location before Build will work. That is, if you move the source files listed in the .MQK file to another subdirectory, QB Plus will no longer be able to locate them for compiling and linking, even if you move the .MQK file along with them. Be sure to change the drive and directory information in the .MQK to reflect the new location(s).

Building the Executable File

Press F10 from the QB Plus Build Window to compile your program and link its modules and libraries together into an executable file in accordance with the Build setup you have specified.

QB Plus will first save your build options list to file, then swap QB/QBX and your program to disk XMS or EMS. It then calls the external build utility program, QBPBUILD.EXE which uses the contents of the .MQK file, BC, and LINK to build your program.

QBPBUILD.EXE first calls BC, passing it the arguments in each of the BC Build lines in turn. BC lines beginning with a single quote are skipped. BC lines with a leading "$" sign cause QBPBUILD.EXE to check the date of the source file against the date of the executable file, if one exists, and skip compilation if the executable is subsequent to the source.

If no compiler errors are detected, QBPBUILD constructs a temporary LINK response file in the current directory from the text following the LINK line. LINK is then called with the Build arguments and the name of the newly created response file. If compiler errors are detected or you placed a single quote at the beginning of the Link line, QBPBUILD skips the link step. The Link step is also skipped when the LINK line and all BC lines begin with a "$" sign, and no source files are compiled because they are all up-to-date with the executable file.

When QBPBUILD finishes it displays the result codes returned by BC and LINK until you press a key. QB Plus then takes over again reloading QB and your source file(s) and Quicklibrary, before returning you to the QB editor.

QB Plus Drives, Directories and Paths

QBPBUILD first uses any path specified as part of the .MQK file for BC and LINK. This lets you use a specific compiler or linker version if you wish. If no path is specified, QBPBUILD uses the BC and LINK programs found in the current directory, or in any subdirectory contained in your PATH environment variable.

QBPBUILD will ignore drives and directories stored in the QB.INI file through QuickBASIC's Options menu. However, as it is an external program, QBPBUILD may be enhanced to recognize these as well as other options you might want to incorporate into the QB Plus .MQK file.

Except as noted above, what you see in the Build window is essentially what you get from QBPBUILD's process.

QB Plus Tips and Traps

1. **Tip:** With a new program, try to load all your modules and save them using the File-Save-All QB menu option before invoking the Build .EXE window. This way QB Plus will fill in most of your option lines for you. Your modules do not need to be full working models to do this—just a single remark in the file and a name will satisfy QB Plus.

2. **Tip:** Have you made major changes to your program and already saved an existing set of QB Plus options, and you don't want to type in all the new module names?

From the DOS command line erase the .MQK file. Then, back in QuickBASIC, pop up QB Plus with all your program modules loaded, and QB Plus will construct a fresh set of "starter options" for you.

3. **Tip:** Use QB Plus if you need to maintain two different executables from the same source files. QuickBASIC will give you one version with the standard options, and QB Plus can give you whatever other version you need. For example:

   a) a real-mode and an OS/2 mode via PDS
   b) an alternate math version and a co-processor version
   c) near and far string versions

4. **Trap:** Do not use a semicolon on the LINK line. If a semicolon is present LINK will ignore the response file and not know what to compile.

5. **Tip:** You can get around QB Plus's 18-module limit by listing more than one object name on each line under LINK. This limit applies only to the number of modules that QB Plus will pull out of the QB.EXE in-memory module list when preparing your "starter set" of BC/LINK options. You are free to add more of your own within the 40 lines and format provided.

6. **Trap:** Although BC will alert you to missing /X/V/W/E switches needed to compile source files containing associated statements, you must remember to add these switches to other modules where you need the /X/V/W or /E object code to also be incorporated.

7. **Trap:** If your loaded Quick Library does not get picked up automatically, or if it is wrongly listed in QB Plus's build window, you may have had a different Quick Library loaded during the session when you saved these settings. Or, if you incorrectly specify the library on the command line and are prompted for the correct library name as QB starts, the corrected name will not be available to QB Plus when it constructs its options automatically.

8. **Tip:** Use the single apostrophe as you would in QB to "comment out" compile or link commands you might not want executed. Placed at the beginning of a BC or LINK line, the single apostrophe will prevent QBPBUILD from executing the line.

9. **Trap:** Blank lines are ignored by QBPBUILD and you can use them to set off parts of your options for readability. However, avoid blank lines below the LINK command, because the position of LINK options is significant.

10. **Tip:** Always save your program before building it, because QB Plus swaps QuickBASIC and your program to XMS, EMS, or disk during the building process. If for some reason QB Plus is unable to recover the swapped information, any unsaved changes you made will be lost. Perhaps more important, this also ensures that the latest version of your program is the one compiled, since QB Plus compiles from the file copy, not memory.

11. **Trap:** QB Plus may not be able to accurately construct a "starter set" of build options if:

    a) You've loaded and unloaded many modules in the current session of QB before invoking the Build .EXE window.

    b) You've specified as the main module, a module that was not the first module loaded.

12. **Trap:** If you have several modules in memory and QB Plus has selected the incorrect module as the main module, you will need to reload all modules into QB being certain to reload the desired main module first. Although you can change the names and the order in which the files appear in the Build window, QB Plus nevertheless saves these setup options under the name of the file it displays as the first BC line. If you've made such changes, then from DOS you should rename the option file to avoid later confusion.

13. **Trap:** While you can usually stop a BC or LINK invocation with Control+C or Control+Break, this will not necessarily cancel the succeeding BC or LINK invocations which are a part of the Build sequence.

External Debugger

The Debugger option causes QB Plus to swap QuickBASIC out of conventional memory and into extended or expanded memory or disk, and then call the program with the name that had been specified in the Change Settings area as the debugger name. This is CV.EXE by default—referring to the Codeview debugger.

Any debugger options and the name of the executable program to debug specified in the Change Settings area are passed as command line arguments to the debugger. If no debug target executable program was entered in the Change Settings area, then QB Plus uses the name of the most recent QuickBASIC program built by QB Plus's Build EXE procedure.

If the debugger cannot be found in the current directory, or in any directory in the PATH= environment list, then an error message will be given.

If no debug file name or a name of a non-existent file is passed to the debugger, then the debugger may give a message if a file is not found at start up. CodeView's message is "Program not found."

If you have no debugger, or wish not to use it in this way, the name of any other executable file may be substituted for the default debugger name in the Change Settings menu. Then, when the debugger option is invoked, the substitute program will run, and be passed the command options and program name as though it were a debugger. This might be an effective way to invoke an external cross-reference, text printing, pre-processor, or other utility.

Macro Keystrokes

The Macro Keystroke feature of QB Plus works with QuickBASIC and QBX to allow recording and playback of keystrokes within the editing environment. This extends the normal editing features to permit the automation of repetitive typing tasks often encountered during program editing.

The recording and playback is not limited to text only, but can extend to QuickBASIC menu selection commands, input to a program running in the QB environment, and to DOS programs shelled to from either QuickBASIC or a running QuickBASIC source program. It can assist not only in editing, but in testing and debugging your program within the environment.

Recording and playback are not supported within QB Plus itself.

QB Plus allows recording of 36 sets of up to 40 keystrokes. A keystroke includes the press and release of the Shift, Control and Alt keys. Mouse activity is not recognized, recorded or played back.

A set of keystrokes may be joined to any other set, such that following playback of the first set, the next specified set automatically plays. This allows up to 1440 (40 times 36) different keystrokes to be recorded and played from a single key.

Recorded keystrokes may be saved and recalled from file; QB Plus defaults to an extension of .KQF. All such files are saved and loaded from the current directory, unless a path is specified as part of the file name itself.

QB Plus is designed around a standard 100% compatible keyboard buffer located at segment 0040h, offset 001Ah in memory. QB Plus's macro record and playback feature may not be compatible with other macro recording and keystroke playback utilities that expand or relocate the standard keyboard buffer.

Recording Keystrokes

To record a series of keystrokes, bring up the QB Plus window and choose the M option for the macro keystrokes. Then select the R option. QB Plus displays a list showing which keys are currently empty,

and which in use. Keys that are joined to one another are denoted by an arrow pointing from the one key to the next. If you pick one of these, playback will automatically include any keystrokes recorded in the key to which your key is joined.

Pick a key into which you wish to record by pressing it. Choose from keys in the ranges A to Z and 0 to 9.

Once you press a key, any existing keystrokes are erased, the menu is cleared from the screen, and record mode is in effect. An "r" appears on QB's top line, followed by the macro key into which recording is being made. As each of your keystrokes are recorded, an ASCII representation of the key code is displayed on QuickBASIC's top line after the Macro Key. ASCII characters are displayed as is; extended keys produce an ASCII representation of their scan codes. The Shift, Alt, and Ctrl keys produce a code both when pressed and released. A special set of codes—not the actual scan codes—are stored by QB Plus for these keys.

Slight delays may be embedded in playback by pressing and releasing an individual control or shift key. This can absorb some playback processing time—useful, for example, between the Run command, and gathering of input by your BASIC program to give QuickBASIC time to get your program up and running before feeding keystrokes to it.

When you have room in the Macro for only four remaining keystrokes, QB Plus sounds an alert after each key press. (If you need more recording room, see the section on joining macro keys.)

To stop recording, press the End Recording key combination. This is Shift+Ctrl by default, but you can change it to Left Shift+Right Shift in the Change Settings menu. This key combination may be shared with the QB Plus menu pop-up command, which likewise may be changed in the Change settings menu or with the /SS command line option.

When the End Recording command is given, QB Plus removes the recording status information from QuickBASIC's top line. In addition, if you are in the QuickBASIC editor at this point and the QB Plus pop-up command and the End Recording are the same, the QB Plus menu window will appear, and you may then view the recorded keystrokes.

Avoid ending your recording on the 40th key. The first key of your End Recording key combination will be saved by QB Plus, and will be the last key played on playback, leaving your keyboard in a possibly confusing shifted or control state, as though the control or shift key is being held down. This can also occur if you happen to halt a playback sequence with the Esc key just after a Control/Shift/Alt key down event was passed to QB.

If, after playing a macro, you find your keyboard responding as though the shift, control or alternate keys are down, simply press and release the control, shift or alternate key, as the case may be, to clear the state.

When using QBX, you should record QBX menu commands by holding down the Alt key and keeping it down while pressing the desired menu letter choice. Only then should you release the Alt key. Although QBX will respond during recording to the physical press and release of the Alt key before the menu letter, it may not work that way on playback. For example, in opening a program file, record the following: Alt-down, F, Alt-up, O, and so forth.

This is not a problem with QB.EXE.

Keystroke Playback

Keystrokes may be played back using one of two methods:

1. **Menu method**

   Select the "P" option from the QB Plus menu, then press the desired Macro key when the Macro Key list is displayed. Of course, you must be in the appropriate QuickBASIC editing context to access the QB Plus menu. Since your keystroke recording always begins in this context, this is often appropriate for playback.

2. **Double-tap**

   If you want to save a step or want to begin playback in a different context than recorded, invoke recording with a double-tap of the play key. The default play key is Caps Lock. To double-tap, press it twice in quick succession. When QB Plus recognizes your double-tap, it displays an "m" on QB's top line, prompting you for your desired Macro key. Press it and playback begins.

During playback, QB Plus displays a "p", the macro key, and the keystrokes being played on QuickBASIC's top line. Press Esc to cancel playback.

Playback differs from recording in one aspect which can sometimes be crucial: timing. QB Plus can feed keystrokes to the keyboard buffer at up to 18 per second with no pause in between, except to wait for the last key to be read. Furthermore, since changes to the Ctrl, Shift, and Alt keys are not buffered, QB Plus delays slightly on these keystroke plays to give QuickBASIC and other applications time to read these changes.

You can take advantage of these built-in delays to insert brief pauses into a key playback sequence to allow QuickBASIC or other programs to complete a process. For example, after pressing F5 from QuickBASIC to run a program, QB takes some time to start the program, then flushes the key buffer to pass a clean buffer to the program. A tap or two of a shift key during recording will, on playback, stall and provide some throwaway keystrokes during this start up process. Each press and release of a shift key inserts about 1/3 second delay.

This is not adequate to span long periods of keyboard inactivity during which a running program is polling the keyboard for certain key codes. For example, although you can record the keystrokes beginning within QuickBASIC to make an .EXE file and then shell to DOS to run it, playback of the recorded key sequence will fail during execution of BC and LINK because they exhaust the recorded keystrokes polling for Control+C and Control+Break. The work-around here is to record separate macros—one to lead up to the start of such a process, and the other you invoke after the process ends to pick up from there.

QB Plus can at other times feed keystrokes to QuickBASIC faster than they can be processed. Also, you might want to slow the playback to better observe or control the process. Use the D option on the QB Plus Change Settings menu to adjust the delay time between keystroke playback.

Viewing Recorded Keystrokes

If playback produces unexpected results, examination of the keystrokes recorded in a key helps in troubleshooting and developing work-arounds.

Press V from the QB Plus Macro menu, and then the desired macro key. The detailed information which then appears shows the number of keystrokes recorded in that key, and the macro key, if any, to which the key is joined. Each keystroke is shown in two formats. The upper is the ASCII representation of the character and scan code for the keystroke. The lower line is the hex value of each of the two bytes recorded for the keystroke.

Extended keys, such as the function and cursor control keys, have zero for the first byte and the extended key code in the second byte. QB Plus assigns special codes from 248 to 255 for the press and release of the Shift, Ctrl and Alt keys.

Saving Recorded Keystrokes

Press "S" from the QB Plus Macro window to save to disk keystrokes that you have recorded in memory. You will be prompted for a file name. Pressing Enter with no file name cancels the save; otherwise, enter a valid DOS file name. QB Plus will add the default extension of .KQF and save your keystrokes accordingly. You may also specify a drive and subdirectory as part of the file name, to save your keystrokes outside of the current subdirectory.

Several editing commands are available during entry of the file name:

Esc	Clear entry, place cursor at left
Ctrl +A	Clear entry, place cursor at left
Ctrl +S	Clear one character to left, move cursor
Backspace	Clear one character to left, move cursor
Ctrl +D	Restore character to right, move cursor
Ctrl +F	Restore all characters, move cursor
Enter	Accept entry as shown

If you should leave QuickBASIC after having recorded but not saved keystrokes, QB Plus will prompt you with a last chance opportunity to do so.

Loading Pre-recorded Keystrokes From Disk

Load a set of keystrokes previously saved in file with the L option on the QB Plus Macro menu window. QB Plus supplies a default file extension of .KQF automatically. Include the drive and directory information for keystroke files as needed for files not in the current subdirectory.

See the Save section above for a list of editing keys available during file name entry. Press enter on a blank file name entry to cancel loading.

You may also specify a keystroke file as part of the QB Plus command line on start up. Use the /KF option along with the file name, in addition to any parameters you wish to pass along to QuickBASIC.

Examples:

QBP /KF	Starts QB Plus and QB.EXE or QBX.EXE using the default key stroke file QBMAC.KQF
QBP /KFdecorate	Starts QB Plus and QB.EXE or QBX.EXE using the keystroke file DECORATE.KQF
QBP /L/KFdemo.wwq	Starts QB Plus and QB.EXE or QBX.EXE using the keystroke file DEMO.WWQ and the default Quick Library QB.QLB or QBX.QLB

If the specified keystroke file cannot be loaded, a message is displayed. Then QB Plus continues to load and start QB.EXE or QBX.EXE.

The supplied prerecorded keystrokes are compatible with QuickBASIC 4.5 and QBX with the default key mapping. QBX users who employ the custom key mapping will need to ensure that any prerecorded keystrokes that are loaded are compatible with the key mapping in use, otherwise keystroke playback may produce incorrect results.

Joining One Set of Keystrokes to Another

You are not limited to playing back just 40 keystrokes at a time. With the Join feature, you can specify another macro to be played when the first macro key finishes play. For example, if you need to record a series of more than 40 keystrokes for a particular operation, you can record the first 40 in macro key A, the second 40 in macro key B, the third 40 in C and so forth. Then join A to B, and B to C. Thereafter, whenever A is called for play, the keystrokes in B and C will automatically play when A has finished. Similarly, whenever B is called for play, you will also get C's keystrokes because of the link between B and C.

Such key joins need not be in alphabetical order—you can jump around as you like. You can also join a key to itself—either directly or through other intervening macro keys—to create an endlessly repeating loop. Note that you can interrupt the playback by pressing with Esc.

For example, if you want to indent every tenth line in your file, you could record 10 repetitions of Down-Arrow, followed by Home and Tab. Then join the macro key to itself.

You can join the keys together at any time—before or after recording. You may have several different macro's which can be replayed individually, or in different combinations or sequences depending on the situation. Join them in one order for one purpose; then remove the connection and rejoin them in another order for a different purpose. The different combinations can be preserved in different .KQF files if you wish.

On the other hand, if you need to record a long series of keystrokes to be replayed from a single key, it is to your advantage to join the keys before recording. With your joins in place, begin recording with the first key in the sequence. When the first key is almost full, it will sound the usual "key full" warning beeps. Continue typing anyway. When the first key is full, recording will continue with the second key in the join list. When that is full, recording will carry into the third key, and so on until the last key is filled.

During each transition you will hear the "key filled" beeps. The display on the screen's top line will tell you which key is being filled so that you will know when you reach the last join key. On that key, the "key filled"

warning beeps will not stop once the key is filled, until you invoke the End Recording key combination. Any keypresses after the last key is filled and the End Recording command will not be recorded.

QB Plus is supplied with a key file named ALLJOIN.KQF, which has 36 empty keys joined consecutively from A to Z to 1 to 9 to 0. You may load this file, record your long macro, then save it under a different name.

Program Execution Profiler

Software optimization means using that combination of design, algorithms and code which best achieves performance objectives. Program execution profiling lets you focus the optimization effort where it has the greatest potential effect. The profiler does not tell you what must be changed or how. What changes are to be made in the design, algorithm, or coding mix is ultimately up to you.

The QB Plus profiler works on the premise that the portion of your program's code where the program spends most of its time is the area where greatest potential gains in performance are likely to lie. Most programs spend more than 80% of their time executing less than 20% of the code. So, rather than waste your time optimizing the startup and exit routines which get executed only once, you're better off to attend to the workhorse code somewhere in the middle of the program that is repeatedly executed.

It may not always be apparent what that workhorse code is from studying the source listings — particularly in a large and complex program. That is where the QB Plus profiler comes in.

QB Plus's profiler allows you to time portions of your program while in the QuickBASIC environment. Although, because of memory considerations, it is not nearly as complete as an external profiler, it can nevertheless give you a very quick and reasonably focused view of where your Basic program is spending its time.

The profiler in QB Plus is designed to report on the execution of your program in several different ways.

First, the total running time of your program is reported. This measure can tell you how much of an overall improvement in your program's execution speed you achieve through your optimization efforts.

Second, it can report what percentage of the program execution time was spent in each of its procedures. As the program executes, QB Plus periodically samples the program's execution location, and stores a code number relating to the SUB or FUNCTION executing at that instant. When the program ends, QB Plus tabulates this sample data and displays

a set of horizontal bar graphs to show the distribution of program execution time across the various routines. Those that constitute the greatest percentage of time thus become the focus of your optimization.

Third, to assist in determining how to optimize, QB Plus accumulates a count for each SUB and FUNCTION call, and optionally displays horizontal bar charts reflecting the percentage of calls by procedure. This can tell you whether the program is spending a high percentage of time in a routine because of a large relative number of calls to it.

Finally, to determine whether a routine's time percentage is the result of a time consuming routine, QB Plus reports on the average duration of each routine's call. This is displayed in a tabular format to the nearest microsecond.

With these reports, you can concentrate on those routines that use the greatest percentage of time. As you make changes, you can immediately profile again within the QB environment to see the effects of the changes, whether it be in relation to reducing the number of calls to a given routine, or in improving a routine's execution speed.

Profiling Set Up

First, load the program you wish to profile. Since the QB Plus profiler collects samples based on procedure calls, the more procedures a program has, the more detailed the information the profiler reports. A program that consists of just a main module, or a main module and a few subprograms will not really tell you where the program time is being spent. However, you can use QB Plus to time the program's total execution time for comparative purposes.

In the QB Plus profiler menu, pressing P toggles the Enabling/Disabling of profiling. Toggle it to Enabled.

Below that, select an appropriate sampling rate with R. If your PC has an AT-compatible real-time clock, the rates range from about 18 to 2048 per second. The faster the rate, the greater will be the precision of the resulting reports but the slower your program will execute. The default rate of 18 gives you a reasonable picture of your program's execution in the initial stages of optimization. If your PC lacks a CMOS clock, then the only available sampling rate is 18.

On the right of the window, the sample buffer should be empty. Before a profiling run is made, you will usually want to press D to discard the samples from any previous profiling run.

Any samples collected from a prior program, or from the current program prior to editing, are not valid and should definitely be discarded. You should also discard any existing samples when changing the sampling rate, as a mixture of samples collected at different sampling rates is invalid. QB Plus does not automatically discard samples, however, in case you want to collect and average samples over several successive profiling runs of the same program. See the section Cumulative Runs for more about this.

Once you are satisfied with the Profiler set up, press Enter and you are now set for profiling. From this point, whenever your program is running QB Plus will sample it. Escape from QB Plus, return to QB, and press the F5 key to run your program and collect samples. Allow it to run for a minute, then end it as you would normally. QB Plus will stop sampling when your program stops, preserving the samples in the sample buffer for you to analyze.

Profiling Analysis

Select A from the QB Plus Profiling menu to see an analysis of the collected sample data. You should do this before making any changes to the program; otherwise the sample data may be invalidated.

QB Plus will examine the sample data and prepare a frequency distribution of the code numbers that have been recorded for each procedure that was executing when QB Plus collected a sample. These code numbers and the percentage each represents are matched with the names of the procedures in memory, and then displayed on a horizontal bar chart.

Bar charts are available for both the routines' percentages of the execution time and the total subroutine calls. All bar graphs are presented in the order in which the subroutines were executed. The time percentage chart is displayed first automatically when the Analyze option is chosen. You can also view the time percentage graph from any of the other Analyze screens by pressing the T key.

Time Percentages

The Time Percentage graph shows the name of the routine, a horizontal bar with a length equivalent to its percentage of the total execution time, and the corresponding numerical percentage. Note that the percentage may or may not add to 100%, and there are routines listed with 0%. This is due to dropping the fractions of percentages in the display.

In examining the percentages, keep in mind that where subroutines are nested, QB Plus will identify only the lowest level routine that is executing at the moment of the sample, and does not identify the caller(s). Therefore, the percentage applies only to time spent by a given routine executing at its own level, not at the level of any routines it may call.

The display can be scrolled up and down to show additional routines if more than a screen-full have been sampled.

Naturally, those routines with the longest bars are the ones with the greatest potential for optimization. You can refer to the two following screens to identify the basis for these routines' time consumption.

Call Percentages

Pressing the C key gives a set of horizontal bar graphs of routine calls. The graphs represent the percentage of the total calls that each subroutine's calls represent. Thus, the graphs give you an immediate indication of which routines were invoked the most heavily. If a routine that takes the highest percentage of program time also represents a high percentage of the calls, then the reason it represents a high percentage of time is likely because it is called frequently.

If a routine that uses a high percentage of time does not represent a high percentage of the calls, then it likely takes a long time to execute relative to the other routines.

Call Duration

Press the D key to display the number of calls to each routine, and the average duration for each call. Those routines which represent a high percentage of time, and also have a long relative duration, are the program's speed bottlenecks. The greatest optimization effect on the

overall program in this case is to speed up these routines. The Duration display can tell you how much speed up has occurred in each routine from one optimization to the next.

These three displays—time percentage, call percentage, and average duration—are what permit you to zero in on what the program's bottlenecks are and why.

Printing the Analysis

Pressing the P key creates a hard copy print out of the complete sampling analysis. Before actually printing, QB Plus tells you the number of lines to be printed, and pauses to let you prepare the printer. Output is sent to the first parallel port, LPT1.

At the top, the printed output contains the name of the program, the sample rate, total number of samples, total number of calls and the program's execution time in seconds. Below that are statistics for each routine covering the time and call percentages, number of calls, and the average duration of the calls in microseconds.

Optimizing

Once the profile analysis tells you where the bottlenecks are, it is up to you to deal with them. Ultimately, you'll have to balance it out to see how far you want to go in optimization.

Your one option, of course, is to do nothing. After trying the executable version and then considering the effort it might take to speed it up, you might conclude the program is fast enough. The use of optimizing compilers, assembler libraries, and such can help improve your odds of such a happy outcome, but these tools can not make up for inefficient design.

Ideally, you want to build your program optimally to begin with. It is best not to wait until the very end of the process to profile and optimize. It may well turn out that more than just some tweaking and fine tuning will be needed. Typically, the biggest performance gains come from significant changes in approach, new algorithms, or changes in the structure or approach to data.

Profile early in the development process, so that tip-offs to potential bottlenecks come early when changes are easier to make.

Using QB Plus with QuickBASIC, this implies the heavy use of FUNCTIONs and SUBs to provide detailed profiling data. That makes sense for other reasons and is fine in the early stages of development—even though these procedure types can be slower than other alternatives. Those routines which ultimately turn out to be time critical can always be converted to in-line code or GOSUB routines later on.

To speeding up a routine you might look to the obvious and simple changes that speed up a QuickBASIC program, and apply them in the time critical areas. This includes using integers, avoiding intensive string handling, and using substitute assembler routines.

Often, though, you will need an order of magnitude improvement in speed to produce a noticeable effect on a program's overall performance. This might call for changes making the program bigger and more complex for the sake of speed in those critical portions. For example, you might find yourself writing extensive initialization and look-up code to precalculate and access time-consuming calculations, or to convert between types so the inner loops will be working with integers, or rearranging a program's sequence so that you can work two separate processes into a joint loop through common data.

Fundamentally, the real value of the profiler is to tell you where the algorithms themselves are inefficient and worthy of improvement.

Profiler Considerations

During sample collection, you want to reduce as much as possible the number of variables between sampling runs, to isolate the results of your changes. That is a good job for the machine. Take advantage of it whenever you can in the sample collection process.

Optimizing, on the other hand, takes thought, creativity and even instinct—sometimes considerable amounts of all three. That is where you come in.

The following information provides further insight into the Profiler's operation, its interaction with QB, and other considerations that will help you interpret results and develop an effective optimizing strategy.

Automating Profile Sampling Runs

Before getting into serious profiling/optimizing cycles, you should consider how you can arrange for the program to replicate its operation and terminate itself entirely on its own, if it does not do that already. You may be able to feed keystrokes to it as discussed below. If not, you should insert loop or control statements into the code where and as needed to cause it to execute in the same way each time you profile it. These statements should cause it to end on its own after having executed the sequence. Once you have completed optimizing a given sequence, you can change the controls to profile some other sequence you need to analyze and possibly optimize.

Profiling and Macro Keys

Programs that gather keyboard input can introduce uncertainties into the profiling process. First, the human response time will vary from one sample run to the next, making it hard to assess the results of interim program changes. Second, the delay while the program waits for the key press could inflate the execution time of the subroutine containing the keyboard input routine, thereby skewing the results.

Recording the keystrokes needed to respond to keyboard input avoids these potential problems. Set up the QB Plus macro recorder and run through the program while recording your keyboard input. Then set up your profile sampling run and execute your program again, but this time use the recorded macro to play the keystrokes back to the program. This way, the keys will be fed to the input routine rapidly and consistently.

Interpreting Changes

The profiling analysis shows the percent of program time spent in each procedure individually, not cumulatively. A calling routine will not be credited with the time spent by another routine it has called. Only the lowest level procedure executing at that time is accounted for during collection of a execution sample.

The implication here is that if you undertake to consolidate a procedure with its callers, you will of course eliminate the procedure and all the time the program spent in it. But you will increase the percentage of time spent in the callers. You need to determine whether the result is a net gain, and the percentage change can be misleading.

Consider a case where SUB1 at 10% calls SUB2 at 40% in a program that took 49 seconds to execute. You eliminate SUB2, putting it as a GOSUB within SUB1 and run another profile. The new profile shows SUB1 at 50% in 45 seconds. SUB1's new percentage is exactly the same as the previous combined percentage for both SUB1 and SUB2, thus it appears there has been no improvement. Yet 4 seconds were saved in program execution time. If the profile runs were identical, the change actually saved 8%.

QB Plus Profiler Overhead

To keep track of program status, QB Plus hooks into BASIC code that executes each line and calls any SUB or FUNCTION of a loaded program. Also, QB Plus intercepts the PC's timer interrupt to handle sampling and other QB Plus functions. Together, these add from two to six percent overhead to BASIC programs running in the environment, an amount which is unnoticeable unless you are timing program execution. Naturally, you will want to allow for this when comparing a program's execution time within QuickBASIC with and without QB Plus loaded, if you need to do this for some reason.

Relatively speaking, the more statements per line a program has, the less the QB Plus overhead effect. Thus, in a small program where you have compressed a substantial number of single statement lines into multi-statement lines, you should confirm that any resulting speed up is intrinsic to the program, rather than through the reduction of QB Plus overhead. You can do that by comparing both the before and after versions of the program in QuickBASIC without QB Plus loaded.

The greatest profiler overhead, however, exists when profiling is enabled and sample collection is occurring. At this time, on each SUB and FUNCTION call and once every sampling interval (18 to 2048 times per second), QB Plus adds sample information in the sample buffer for the currently executing routine. This overhead can be considerable, particularly where a large number of routines exist in program. You should disable sampling when you want your program to run at the normal QuickBASIC environment speed.

Potential Sampling Errors—Halt/Resume Sequence

Although you can halt and resume a program while it is being sampled, the results could be inaccurate under certain circumstances.

QB Plus starts timing a program when one of its program lines begins to execute. QB Plus then stops timing when QuickBASIC returns to the editor environment. Accordingly, program execution time can be understated when the halt occurs within a multi-statement line or a basic intrinsic function, and on program resumption there is a passage of time before the next line is executed. Consider the following program fragment:

```
A$ = "" : A$ = INPUT$(2)
PRINT "The contents of A$ is: ";A$
```

If you were to halt the program while it is executing the input statement, on resumption none of the program time occurring during the input statement would be sampled until a key is pressed and the next program line begins executing. Either allow for this in your timing, or (preferably) avoid breaking into the program during profiling.

Potential Sampling Errors—System Timer Reprogramming

At a sampling rate of 18 per second, QB Plus depends upon the PC timer tick—which occurs 18.2 times per second—to know when to interrupt your program to take samples. BASIC programs, Quick Library routines and BASIC statements that alter the timer tick rate could throw QB Plus's profile timings off. Consider using a different sampling rate, if available, since the real-time clock on which the higher sampling rates are based is rarely modified by programs, and should therefore be more reliable.

Sample Rate Harmonics

Any subroutine that is invoked at the same or multiple of the sampling rate could end up either under- or over-represented in the sampling. For example, a process occurring exactly in sync with the timer tick will either never be sampled, or will be the only process sampled.

Similarly, routines called repeatedly at fixed intervals may suffer from stroboscopic effects of their call frequency combined with the sampling frequency to distort their actual execution duration. Very short routines called in a tight regular loop are especially susceptible to his effect.

Particularly on your first profile of a program, try several different sampling rates and compare them. If you do not have multiple sampling rates available on your PC, try opening a watch variable in the QB debug menu, to slow your program down with respect to the fixed sampling rate. This will help to identify stroboscopic or harmonic effects.

QuickBASIC History, Watches, and Breakpoints

Since these will affect a program's execution speed, it is usually better to disable all of them during profiling—except perhaps as noted in the section about Sample Rate Harmonics. If you do have any of these in effect, at the minimum be sure not to change any of their settings between sampling runs you wish to compare.

Cumulative Sampling Runs

Run times and sample proportions naturally vary slightly from one profiling run to the next, even where no changes have been made to the program. You may thus wish to average the results of several runs. This may be done simply by not discarding the samples between successive profiling runs. The analysis then will cover all the samples from all the runs. The elapsed time and call counts will reflect the combined time and calls, so you will have to divide this by the number of runs to get the averages per run.

QB Environment Versus Executable Programs

There is a high correlation between profiling results obtained for a program run in the environment and the same program run as an executable program from the DOS command line. There are differences, though, about which you should be aware if your ultimate objective is to optimize the executable program:

1. **.EXE programs generally run faster**

   There are a variety of reasons discussed later for this. Before undertaking major revisions in your code based on profile results in the environment, try out the program in its executable form. Maybe it is fast enough already. At the minimum, set up some timing tests that you can later use as a benchmark to be certain that changes made to speed it up in the environment have the desired effect on its executable version.

You might be able to use recorded QB Plus macros here too, if need be, to get consistent timings through a user interface. To do this, you will have to first record the keystrokes while in the environment—you may need to tip toe around QuickBASIC for certain keystrokes. Then use the QB Plus TIMERUN.EXE add-on program to run the executable version of your program. Run it through QB Plus's Switch Program feature with the name of your compiled stand alone .EXE version as the command line argument. When TIMERUN prompts for a keypress to begin timing, hit your macro key. TIMERUN will show the execution time in ticks after your program ends.

2. **Added overhead in SUB/FUNCTION calls in the environment**

Generally, there is considerably more overhead in procedure calls in the environment than in an executable version of the same program. Further overhead is added by QB Plus's profile sampling. For this reason, eliminating SUB and FUNCTION calls by replacing these routines with in-line code or GOSUB routines can significantly speed up a program running in the environment. It will also speed up the executable, but not by nearly as much.

Thus, don't be misled by performance gains in the environment in this regard. Avoid the temptation toward wholesale collapsing of your code into one flat main module in the interests of optimization. The advantages that code and data encapsulation provided by SUBS and FUNCTIONS have is well worth it for the 80% of your code that is executed only 20% of the time.

3. **Added overhead in Quick Library calls in the environment**

Replacing a simple BASIC routine with an equivalent written in assembly language may often not lead to the level of performance improvement in the environment you might expect. This is because the overhead in the call to the routine may represent a substantial portion of the call time of the routine. The assembler substitute may well have a significant effect on the compiled executable, however, particularly if it replaces a call to a time-consuming portion of the BASIC runtime library. You can determine this only by testing the executable.

The best Quick Library replacement for both environment and executables is an assembly language routine that integrates several low-level functions into one call, thus reducing the overhead of multiple calls.

4. **In-line code optimization by the BC compiler**

   The environment has to be able to reconstitute your source code for display and file storage. The BC compiler, on the other hand can condense things and does. If the routine which appears to be a bottleneck in the environment can be condensed down to in-line code by the compiler, the routine might not be a bottleneck for the executable.

   Consider this somewhat over-simplified example:

```
A = A * 2 + 8 + 10
```

   The compiler can boil this down to a few assembly lines, because it consolidates 8 + 10 into 18. However, the environment will require dozens of instructions to process this, because it cannot combine steps. Gains you make optimizing this statement in the environment will not necessarily pay off to the same extent in the executable.

Profiler Capacity

The samples collected by the profiler are all stored in main memory, which imposes certain limits on the number of samples which can be collected.

	Number	Approximate Duration
Total Procedures	280	
Total Calls	2,147,483,647	30 hrs. minimum
Maximum calls per routine	2,147,483,647	30 hrs. minimum
Total Time Samples	2,147,483,647	291 hrs. minimum
Maximum samples per routine	2,147,483,647	291 hrs. minimum

If the number of routines executed by a program during sampling exceeds the total capacity, sample data will not be collected for the excess calls, the graphic displays will show a warning message, and the results may be unreliable. In this event, you should reduce either the overall sampling duration or the scope of the program sampled. This

can be accomplished by inserting control statements into your program to cause it to bypass portions. You could also simply terminate the program short of the routine sampling limit.

Memory Viewer

QB Plus's Memory Viewer allows the QuickBASIC programmer to inspect the contents of almost any memory location. As such, it represents an extension to the built-in QuickBASIC Immediate and Watch windows allowing you to examine data at its most fundamental level.

QB Plus provides the following features:

- Display of memory by byte, by word (unsigned integer), by signed decimal integer, by long integer, by vectors (far pointers), and by ASCII characters.
- Access to your PC's hardware I/O ports, and to the CMOS machine configuration area.
- Expanded memory support for machines with expanded memory and a LIM EMS 4.0 or higher Expanded Memory Device Driver.
- Extended memory support, via protected linear addressed mode, for PC's equipped with extended memory.
- Additional extended memory access via the XMS 2.0 or higher specification if a conforming XMS device driver is installed.
- A "Real time" mode provides continual on-screen updating of memory contents giving a dynamic display of changing memory values.

Caution and Limitations

Caution: I/O Ports:

QB Plus lets you view the contents of both memory addresses and I/O ports. Unlike memory which is merely a data storage area, some I/O ports actually control electrical switches that are turned off or on when accessed—usually by the CPU outputting a particular value to the port. In rare cases, depending upon the electrical design, a port's hardware switch can be thrown when reading the value at that port.

In Port View mode, QB Plus reads from the port in order to display the value on screen. While this is not a problem for most IBM and compatible PCs with the standard peripherals, there is a remote possibility for a problem where a PC or peripheral card responds to a port read as

well as write. Since ports control important PC components — including video, keyboard, timer chips, DMA lines, floppy and fixed disk drives—you should confirm that port reads will not be harmful to your system before invoking the Port View mode of QB Plus.

By default Port View is disabled. You must start QB Plus with the /PV command line option, or enable it in the Change Settings area, in order to display I/O port information.

Hardware Limitations

The primary purpose of the QB Plus memory viewer is to display the contents of addressable memory. Addressable memory is one megabyte for 8088 CPUs, 16 megabytes for 80286s, and four gigabytes for 80386 and 80486 PCs. QB Plus Version can display addressable memory up to 16 megabytes. Memory on peripheral adapter cards (other than EMS memory), on support chips, or even within the CPU itself, cannot be viewed if it does not appear to the CPU to be within the conventional address space.

Although QB Plus will work on most IBM and compatible machines, the fact that it performs a number of its tasks in direct interaction with the hardware means that some functions may not work properly or at all on some clones or with certain hardware and software combinations. If you are not getting the results you expect, first refer to the features descriptions which follow or the supplied README.DOC file, for more information about specific known incompatibilities. If that does not resolve it, please report the problem to Crescent Software.

Memory View Screen

The Memory View screen is first presented whenever the V option is chosen from the QB Plus main pop up window. This is the primary memory examining screen.

On the left side is a list of memory addresses in hexadecimal notation. To the right, and separated from the addresses by a double vertical bar, are the memory contents, also in hex. In the Byte viewing mode (to which QB Plus defaults the first time it is popped up), the ASCII representation of the memory contents is additionally displayed along the right side of the pop up window.

The address section of each line represents the starting memory address for the memory values appearing to the right on that line. The default is the standard segment:offset addressing style with which conventional memory below the one megabyte limit is addressed by the CPU in real mode.

To afford access to memory above the one megabyte conventional limit, QB Plus provides an additional address component: Huge. The Huge value is a multiple of one megabyte. A Huge value of zero refers to conventional memory, and unless your computer is equipped with extended memory this value is always zero. Refer to the section that discusses linear extended memory access for more details about how QB Plus handles Huge segment:offset addressing.

Below the 18 address and memory content screen lines is the display status line showing the current address and viewing mode. This line is also used to input specific address values you want to view.

QB Plus defaults to Real Time display of memory. This means that the screen is continuously updated, to reflect changes to the memory being displayed.

Getting Help

The lower border of the QB Plus pop up window contains a prompt line that shows the available commands. If you press F1 QB Plus displays a screen containing more extensive help information.

Getting Around in Memory

While viewing memory contents, you can move from one address to another either by specifying the starting address in hex, or by using address pointer movement keys.

In either case, the starting address is the determining factor for the display. All screen displays begin with the starting address on the first line. The default is a multiple of 16 bytes, ie 0000h, 0010h, 0020h, etc. You may specify a start point which is not on a 16-byte multiple using the O (Offset) command. The 16-byte multiple, however, will be restored when the segment boundary is reached.

Each successive line moves upward in memory by 48 bytes in ASCII mode and by 16 bytes in all others. When the end of a segment is reached, addresses are rolled over to the next higher segment, or wrapped around to the beginning segment, depending on the address mode and type of memory being addressed.

A starting address always consists of three components: Huge, Segment, and Offset. Each must be specified separately either by entering the individual value, or with the movement key for that component. While the offset component will always refer to an individual one-byte address, the segment and Huge components take on different meanings depending upon the address mode in effect—linear, EMS, or XMS. If your PC is equipped only with conventional memory, the huge component will always be zero. See the respective portion of the Address Mode section for details on segments, segment equivalents, Huge, and handle values as they apply to each mode.

Movement Keys

The left and right arrows move the starting address one line at a time, while the up and down arrows move a screen at a time. PgUp and PgDn decrease or increase the address one segment size, or its equivalent, at a time. The < and > keys move the huge component one increment back or ahead.

The Home key jumps to the first offset in a segment or equivalent, while the End key jumps to the last.

Entering Addresses

Press the O key to input a specific offset address in hexadecimal notation. The input values must be between 0 and FFFFh.

Similarly, S prompts you for a segment or equivalent value (meaning page, when viewing EMS). Pressing H prompts for a Huge or Handle value. If you have no extended or expanded memory, the huge value is meaningless, and pressing the H key has no effect.

If you change your mind about entering a value, press Enter on an empty input line to cancel. The backspace and arrow keys may be used for editing.

Out of Range Addresses

If the values you use produce an address that does not have memory associated with it, you may see a series of FFh values on the line for that address. Some PC's will produce random values and the Real Time mode may exhibit a display of constantly changing values.

In XMS or EMS mode, addresses out of the range allocated to a handle will produce blank lines.

Viewing Modes

Byte

In Byte mode, memory is displayed one byte at a time. That is, on each line following the address are sixteen hex values representing the 16 bytes beginning at that line's address. The bytes are displayed in the same order they occur in memory.

Since each hex byte consists of two characters, you can determine the memory address of a given byte by counting every other character beginning from the left. In the example below, the left-most byte, 8, is at address 0000:0300. The next byte to the right, &H84, is at 0000:0301; byte value &H4E is at 0000:0302.

To simplify counting, the first eight bytes are separated from the second eight bytes by a space. Thus, the first byte to the right of the empty display column in the middle is always eight bytes beyond the starting address for the line. In this example line, the value at address 0000:0308 is &H31:

```
0000:0300   08844E59393E3315 31232425262F2F2F  .aNY93.1#$%&///
```

In byte mode, the ASCII characters for each byte are displayed at the right. Non-displayable control characters (ASCII values less than 32) are represented by periods.

Word

In Word mode, memory contents are treated as a series of words. A word is the same as a 16-bit integer: two consecutive bytes.

On each line, QB Plus displays consecutive words beginning at that line's memory address. Each two-byte word is separated by a space from its neighbor. To determine a given word's memory address, regard the left most word as zero, then count each word to the right until you reach the word you are interested in. Multiply its count by two and add the result to the line's beginning address.

In the example below, the word value at address 0000:0306 is &H1533.

```
0000:0300   8408 594E 3E39 1533 2331 2524 2F26 2F2F
```

Be careful not to apply this technique to the byte components of the words, however. For example, while &H1533 is at address 0000:0306, this word's first byte, 15h is not at 0000:0306 as you might expect. In reality, it's the lower byte of the word that appears first in memory—&H33 in this case. In other words, the lower byte is always at the lower address.

If you want to locate the address of an individual byte, you are better off using the byte mode formatted display. Compare the two examples below — the first in byte mode; the second in word — and notice the reversal of adjacent byte pairs.

Byte:

```
0000:0300   08844E59393E3315 31232425262F2F2F  .ANY93.1#$%&///
```

Word:

```
0000:0300   8408 594E 3E39 1533 2331 2524 2F26 2F2F
```

Since each display line defaults to a 16-byte multiple, each word displayed by QB Plus begins on an even numbered byte. Words, however, do not have to do this. A word can start at an odd byte address, but when it does it forces extra processing for the CPU. It is thus advantageous to have words aligned on even addresses. QB Plus defaults to that assumption.

If you wish to view a word or words that begin on an odd byte address, use to O command to enter that address. All the words on the display will now be based on low order bytes beginning at odd addresses, and naturally their values will change.

Note that the odd, non-16-byte multiple, used as the line starting address remains in effect only when the screen display does not cross a segment boundary. This is because the CPU cannot create a word from bytes in two separate segments, but instead wraps the offset value to zero. Thus, when the boundary is crossed, QB Plus adjusts it to even bytes by adding or subtracting one byte, and the line containing the boundary crossing has invalid or missing data from that point to the end of the line.

Integer

BASIC does not have a built-in variable type of Word. BASIC's closest equivalent is the Integer—a 16-bit signed value as opposed to the 16-bit unsigned Word. Basic treats the highest bit as a sign bit. If the bit is set, the value is negative, otherwise it is positive. Therefore, the smallest integer is 8000h (-32768) and the largest is 7FFFh (+32767). Any value that has a most significant digit of 8 or more is considered negative by BASIC.

A hexadecimal display of type Integer therefore looks identical to a hexadecimal display of type Word. QB Plus provides the Integer values in signed decimal. Note the two lines below that show the memory values in Word and Integer formats as used by QB Plus.

Word:

```
0000:0300   8408 594E 3E39 1533 2331 2524 2F26 2F2F
```

Integer:

```
0000:0300  -31736 22868 15929 5427 9009 9508 12070 12079
```

Address and segment boundary considerations for Integers are understandably the same as for Words.

Long

Longs are 32-bit integers, or double-words. They are stored in memory with low-order word first, then the high-order word. Each word is displayed in a reversed order fashion as well, low-order byte leading the high-order byte.

QB Plus automatically reverses the memory storage order for the Long display, so that you are presented with a 32-bit long integer with the bits in descending order from the left.

Determining a given long integer's address is similar to determining a word's address. Starting from the left with the first value as 0, count right to your long integer, multiply by four, and add the result to the address at the beginning of the line.

In the example of the Long format below, the long integer &H15333E39 is at address 0000:0304.

Because of the storage order reversal of the bytes and words, be careful not to assign memory addresses of the component parts of a long integer in the order they appear on the display. It is wise to switch to the appropriate byte or word mode format if you need to determine the address of a specific byte or word. Compare the examples below:

Byte:

```
0000:0300   08844E59393E3315 31232425262F2F2F  .ANY93.1#$%&///
```

Word:

```
0000:0300   8408 594E 3E39 1533 2331 2524 2F26 2F2F
```

Long:

```
0000:0300   594E8408 15333E39 25242331 2F2F2F26
```

Note that for purposes of this display mode, QB Plus assumes the long integers all line up next to one another on a 16 byte address multiple. In reality, this may or may not be the way the values in this range of memory are actually accessed by programs. For example, BASIC can and does store long integers starting at any even-numbered address. Use the Offset value entry to specify a starting line address other than the 16-byte multiple. See the discussion in the Word viewing mode earlier for more on odd-byte addressing, and reversion to even addressing at segment boundaries.

Vector

The addresses of memory addresses may themselves be stored in memory. In Vector mode, QB Plus displays these address references as segment and offsets joined by a colon (:). Each segment:offset combination is separated by a space from its neighbor on the display.

Addresses of addresses may variously be referred to as pointers or vectors. Pointers which are 16-bits wide are known as near pointers, and they represent the offset portion of a given address. Near pointers refer to a memory location with a 64K segment whose value is assumed.

Far pointers are 32-bit addresses that contain both the segment and offset values, enabling address specification anywhere in a one megabyte range. In the first 1K of the PC's address space are 256 such pointers, known collectively as the Interrupt Vector Table. Each 32-bit segment:offset vector points to the beginning of an interrupt service routine elsewhere in memory.

QB Plus permits display of memory addresses as though they were a series of such vectors, aligned on a 16-byte address multiple, just like the Interrupt Vector Table. This may or may not be how this memory is actually treated by the programs which access it. Use the Offset input command to specify a different starting address appropriate to the vectors you wish to display. See the discussion in the Word addressing section for cautions on values that cross segment boundaries.

Note that, like the long integers described above, the actual byte components of the segment:offset addresses are not physically in memory in the same order they appear on the display. The low-order, offset portion of the vector precedes the segment portion. Within segment and offset values, the low bytes lead the high bytes. The byte, word and vector examples below illustrate this in context.

Byte:

```
0000:0300   08844E59393E3315 31232425262F2F2F  .ANY93.1#$%&///
```

Word:

```
0000:0300   8408 594E 3E39 1533 2331 2524 2F26 2F2F
```

Vector:

```
0000:0300   594E:8408 1533:3E39 2524:2331 2F2F:2F26
```

ASCII

ASCII display mode presents memory contents as though it were ordinary text, 48 characters to the line. A dot (.) substitutes for control characters—characters with ASCII values below 20h.

Ports (Caution)

I/O ports are a type of memory that can store and provide information passed from the peripherals connected to the ports. QB Plus permits display of data from up to 65536 port addresses, although standard PC's are equipped with only 16384 possible port addressees. The port data as displayed by QB Plus thus repeats at 16K intervals.

CMOS

QB Plus shows the CMOS hardware configuration area for PC's so equipped—typically those compatible with the IBM PC-AT and later. 100% IBM compatible machines allow reading the CMOS area by first writing the value of the CMOS register to be accessed to port &H70. That register is then made available by the CMOS circuitry at port address &H71. Although the standard PC has only a total of 64 CMOS registers, QB Plus displays 65536 CMOS addresses with the data repeating at 64 byte intervals.

QB Plus does not detect if your machine has no CMOS, or that the CMOS is incompatible. In CMOS mode, QB Plus will simply and repeatedly write to port &H70 and read from &H71, displaying the results which, if there is no CMOS, are unpredictable.

Real Time

Real time means that the screen is continuously updated with changes to the memory being displayed as they occur. For a demonstration of this feature, look at the memory addresses beginning at segment 0000 and offset 0400. You will be able to see the timer count changing in response to your computer's clock ticks, and the changes to the keyboard buffer and keyboard flags in response to your key presses.

This real time feature is useful in watching what is happening with other types of interrupt service routines (like the timer tick or keyboard handler that leave memory changes in their wake), or the activity or status at a serial or parallel port.

Using the Real time mode adds delays to the keyboard response, and you can disable it to speed up scrolling through memory using the R key.

Getting Information

The F3 key provides general information on conventional, and expanded and extended memory.

Conventional Memory Information

On the first line is your PC's total installed conventional DOS memory. The is the figure reported by the Get Memory Size function of your computer's BIOS (Interrupt 12h). This figure may not entirely reflect the actual conventional memory used by DOS, if for example, your PC has memory above the 640K range managed by certain types of memory device drivers.

Conventional memory locations used by QB Plus are also listed in this panel. The QB Plus's screen save area contains the underlying QuickBASIC screen. The View Buffer is used by QB Plus to swap expanded or extended memory into conventional memory for display purposes. Below the Screen save area is QB Plus's stack. Examining these areas of memory may produce confusing results as the memory may change in the process of viewing it, or may appear to duplicate memory elsewhere.

Below that are memory addresses associated with QuickBASIC. The data segment is where QB stores your near variables—the default segment set with DEF SEG.

Expanded Memory Information

If you have expanded memory and an expanded memory device driver that conforms to the LIM EMS 4.0 specification, information about the driver and the memory it manages will appear. Information provided includes the driver's version number, the segment address of the page frame, the size of EMS pages—both standard and raw—and the total number of EMS handles along with the number in use. Available and allocated EMS memory is also shown. See the expanded memory portion of the addressing mode section for further details on expanded memory.

On a subsequent page which is called by pressing the H key, you may obtain a list of the existing EMS handles and amount of memory currently allocated to each. The handle number is given in hexadecimal, while the memory size is in decimal Kilobytes. If you have an XMS

device driver concurrently installed, both EMS and XMS handles will be shown. In that case, each EMS handle will be identified with a small e and the XMS handles with a small x.

Extended Memory Information

If you have extended memory, the amount of installed extended memory, taken from your PC's CMOS configuration area, will be shown, followed by the available extended memory figure reported by your PC's BIOS Get Extended Memory Size function (Interrupt 15h, function 87h).

These two figures may differ where a program or device driver has allocated extended memory for itself. In order to protect the memory it has taken, usually from the end of the installed extended memory, the program or device driver will capture the 15h interrupt and report a value which has had its share of extended memory deducted. Thus the CMOS memory size may be greater than the size reported by interrupt 15h.

Accurate reporting of these figures depends highly on machine register and BIOS compatibility with the IBM PC-AT. If QB Plus's total extended memory figure is wrong, your machine's CMOS is structured differently or in a different location than QB Plus expects. On some PC's, this figure may reflect only the extended memory contained on the system board or memory card, and not the additional memory contained on an add-on peripheral card installed in one of the standard slots. This alone should not prevent QB Plus from successfully displaying extended memory information, however, since QB Plus does not rely on this value.

On the other hand, if you know for certain that the "amount available" figure reported by QB Plus is wrong—it's far larger than the amount of extended memory you have, for example—then there is some incompatibility in the BIOS. In this situation, until you can confirm the accuracy, you should regard as questionable any extended memory data displayed by QB Plus that is not obtained via an XMS driver. See the XMS section of Address Modes for more about XMS.

If you have an XMS driver installed that conforms to at least the XMS 2.0 specification, information on the memory managed by the XMS driver, the number of handles available, and the specification level and internal version level of the XMS driver are shown.

EMS/XMS Handle List

On a subsequent page, a list of handles is provided. At the <Handles> <OK> prompt press Enter or Esc to skip the handle listing and return to the memory view screen, or press H to see the handle list.

The handle number is shown in hexadecimal, while the memory allocated to it is in decimal Kilobytes. If you have an EMS device driver concurrently active, its handle list will appear first, with each EMS handle identified with a small e following the handle number. XMS handles are distinguished by an x. There may be a slight delay when displaying XMS handles, while QB Plus interrogates the XMS driver on the validity of each of the 65536 possible handle numbers.

Not all of extended memory is allocated by handle, however. In the XMS memory standard, handles are assigned to memory blocks allocated from extended memory above the first 64k. Such memory can be accessed only in protected mode.

The first 64K of extended memory, referred to as the High Memory Area or HMA, can also be accessed in real mode and used by a program just like it would use conventional memory. If this memory area has been allocated to an application, it will be reported as being in use. Otherwise the HMA size in Kilobytes (usually 64K), is shown.

Addressing Modes

Conventional Addressing

The conventional address mode is the default. On machines without expanded or extended memory, this is the only available address mode.

The addresses are given in three-parts: Huge, Segment and Offset. The Huge component has no meaning on computers without extended memory, and will always have a value of zero. Otherwise, the Huge value represents a one megabyte block, with zero as the block beginning at address zero, one as the block beginning at one megabyte, and so forth.

Segment, of course, refers to the traditional overlapping 64k blocks spaced 16 bytes apart—in this case within the one megabyte block denoted by the Huge value. The offset is the byte count from the start of the segment.

Addresses are accessible from segment 0000h, offset 0000h to segment F000h, offset FFFFh, or any equivalent. Scrolling the display beyond the one megabyte range will wrap the addressing back around to zero. However, in ASCII mode, or if offsets are not on a 16-byte address multiple, the values on a line after the one megabyte boundary is crossed are taken from a zero offset within the segment. This emulates the way the CPU would handle the wrap around to zero of the offset value given a fixed segment value.

The arrow keys primarily affect the offset component of the memory address. Use the left and right arrow keys to scroll forward or back through memory one screen line at a time. In the Byte, Word, Long, and Vector modes, one line encompasses 16 bytes, while ASCII is 48 bytes. The up and down arrow keys move you one screen full of memory data at a time.

The Home key takes you to the first offset address within the starting segment displayed on the top line. The End key positions the starting display address of the first line such that the last line of the display shows the last bytes in the segment.

With the PgUp and PgDn keys, memory addresses are reduced or increased one segment size at a time.

When used with a PC equipped with extended memory, the less-than (<) and greater-than (>) keys move forward or back one Huge value—representing one megabyte—at a time. For Huge values greater than zero, your PC must support extended memory moves using the BIOS interrupt 15h, function 87h. If your PC has extended memory installed but hangs up or otherwise acts erratically when the QB Plus memory viewer is invoked, or when a huge value greater than 0 is requested, you may need to disable QB Plus's direct access to extended memory. Specify the "/NX" option to QB Plus at start up.

Press the "H", "S", or "O" keys to enter a specific Huge, Segment or Offset value in hex.

Linear Addressing

For computers that are equipped with extended memory, all displayed values are first moved via protected mode to a display buffer in conventional memory. The semicolon (;) key permits you to toggle between

the segmented addressing scheme described above under conventional mode, and a linear address scheme that treats all of your PC's memory up to 16 megabytes as one giant block.

The linear address shown is the single 24-bit value that is used in protected mode to retrieve the values you see. The Huge/segment:offset references are first converted to linear address equivalents. For example, a Huge value of 1 would refer to the first megabyte of extended memory beginning at address 100000 hex, or 1048576 decimal. The segment:offset portions of such an address would fall roughly within the first megabyte of extended memory.

The address movement keys described in the conventional mode above work in an equivalent way in linear mode. The left and right arrow keys move a line equivalent, and the up and down arrows move a screen equivalent. PgUp and PgDn move by a 64K block, and the greater- and less-than keys move by one megabyte.

As with Huge mode, to use the Linear mode your PC must support extended memory moves using the BIOS interrupt 15h, function 87h. If your PC has extended memory installed but hangs up or otherwise acts erratically when the QB Plus memory viewer is invoked, or when a huge value greater than zero is requested, you may need to disable QB Plus's direct access to extended memory. Specify the "/NX" option to QB Plus at start up, and Linear mode will be disabled. In this case, conventional mode with a Huge value limit of zero will be used instead. You will need an XMS driver, or a EMS driver emulator to view extended memory in this event.

Expanded Memory Addressing

If your PC has expanded memory (EMS) and an EMS driver conforming to the Lotus-Intel-Microsoft (LIM) EMS specification 4.0 or later, QB Plus will provide for viewing EMS memory contents.

Pressing "E" toggles between EMS and Conventional or Linear modes.

With EMS, the Huge/Segment:Offset addressing takes on slightly different characteristics than in Conventional/Linear modes. EMS memory is not accessed directly; rather, it is viewed through a page frame located in conventional memory. When an application requests

access to expanded memory through the EMS driver, the driver makes portions of that memory available through the conventional memory page frame.

There are a number of variables associated with this method. EMS memory is first allocated to a requesting program in 16K multiples, called pages, which are associated with a unique handle number provided back to the requesting program. The number of pages per handle varies depending on the number requested and available. Handle numbers may range from 0 to FFFFh. Even the page size can be altered from the 16k default, but, fortunately it rarely is. The EMS driver can be asked to report on all of these variables, and QB Plus uses this information to keep EMS addresses within meaningful ranges.

When displaying EMS memory, QB Plus uses the Huge value to represent the handle. Only EMS handles that have been allocated contain valid data. Thus, when an unallocated handle is requested for viewing, QB Plus displays a blank memory value area.

QB Plus uses the segment column value to represent pages allocated to a given EMS handle. Use the "S" command to specify a desired page number, or the PgUp/Dn keys to move one at a time. Non-existent page values may be specified with the "S" command, but they will return blank data; scrolling past the number of pages allocated to a handle wraps the page counter back to 0.

The offset value represents the byte address within the EMS page, with a limit of 16k (3FFFh).

EMS handles may be named in addition to being numbered. Naming is up to the application. If there is a name associated with the handle, QB Plus displays it following the EMS indicator on the bottom line below the memory value lines.

XMS Memory Addressing

If a driver that supports Extended Memory Specification (XMS) version 2.0 or later, such as HIMEM.SYS, is loaded, pressing the X key will toggle between XMS and conventional linear mode. (The latest version of HIMEM.SYS, along with the XMS specification, can be obtained directly from Microsoft.)

Like EMS, XMS uses a block allocation scheme built around handles to let multiple programs share memory in a cooperative manner. Since XMS runs only on PC's with 80286 or later processors, a larger number of memory locations may be addressed than with EMS, which must be compatible with 8088 CPU's. Accordingly, the XMS block size for a given handle is variable between 1K and 16 megabytes. Offsets within the block are 24-bit linear values.

QB Plus uses the Huge address component to represent the handle number, while the segment and offset components are used to form a 24-bit linear address. Thus the movement keys, and the H, S, and O commands adjust viewing addresses in a manner similar to linear mode. Invalid handles and unallocated addresses within valid handles produce blanks where the memory values would otherwise appear.

XMS also manages the first 64K of extended memory, which is uniquely addressable by the CPU in both protected and real mode. A program that uses this address space, known as the High Memory Area or HMA, can access this area with its code, data, or both, almost as if it were conventional memory. Within certain limitations, the XMS driver will allocate this space in its entirety to the first program that asks. No handle is provided since the HMA area is always at the same address; other programs are informed when requesting the HMA that it is already in use.

Whether the contents of HMA may be examined by QB Plus depends upon the XMS driver and the class of PC. QB Plus uses the XMS Move function to copy a block of extended memory from a handle/offset location to a conventional memory display area. As the HMA is not accessed by handle, nor by the XMS copy block function, the HMA addresses are not accessible with QB Plus's XMS mode. Instead, this area must be viewed in either conventional or linear mode.

However, if your PC does not support the BIOS interrupt 15h, function 87h extended memory moves, this area cannot be accessed directly. Even so, XMS drivers running on 80386 and 80486 are capable of mapping memory in and out of the HMA area. Thus, it is possible that an application can write to the HMA, then release HMA, and HMA will appear non-existent to QB Plus.

Interestingly, 80386 extended memory managers that use extended memory to emulate EMS may allocate an EMS handle to the HMA. If so, and you can identify which EMS handle that is, you can view the contents of the HMA in the EMS mode.

Because of these complications and the additional resident memory working around them would require, QB Plus version 1.0 does not support XMS-supported viewing of the HMA.

The Program Switcher and Add-On Accessories

The QB Plus Switcher lets you run another program from within QuickBASIC, yet allows the other program almost all of the normal conventional address space as though QuickBASIC were not loaded. This makes it possible to quickly switch between QuickBASIC and other large applications and environments, without unloading and reloading QB/QBX, your source files, and Quick Libraries.

You can use this feature to invoke external add-on accessories to QB Plus that you write yourself in QuickBASIC and compile to an .EXE file. The Switch Program menu lets you choose from among executable files in the current directory by using a QB-style Pick list. Once selected, a couple of keystrokes from the main QB Plus menu invokes the external program.

Invoke the switcher by pressing S from the main QB Plus menu.

On the next screen, the Tab and Shift+Tab keys permit movement between input boxes similar to QuickBASIC. These input boxes accept the name—including drive and subdirectory—and any command line arguments for the new program that is to replace QuickBASIC. If only the filename is given, QB Plus will assume an .EXE extension and search in the current directory, or along the directory list specified in the "PATH=" environment variable. If no such file is found, a message to that effect is displayed. If the desired file is a .COM file, then that extension must be given explicitly as part of the filename entered.

As an alternative to entering the file name, you can use Tab or Shift+Tab to move to the top box, where a listing of up to the first 100 .EXE files in the current directory is displayed. This is similar to the familiar file picker box in QuickBASIC. Use the arrow keys, Home, End, or the first letter of the name to move the cursor to the file name of choice, and press Enter. That file name will then be placed into the file name box below.

Once you are satisfied with the file name and command line arguments, move to the OK button at the bottom of the window and press Enter. Pressing Esc at any point before this cancels everything.

QB Plus will search for the specified file and report in the upper box whether or not it was found. If not found, QB Plus pauses briefly before returning. Otherwise, it swaps QuickBASIC out of memory, preserves certain machine settings, and runs the specified program. When finished, QB Plus recovers QuickBASIC, restores the prior machine settings, and returns to the QB Plus main menu. Any errors are reported in the upper box.

Although the swapping process happens quickly, there is a lot that goes on by way of preparation and recovery from the swap. If certain hardware or software errors occur in the process, it may be impossible to return QuickBASIC to operation, and QB Plus will have to terminate itself as well. You should therefore save any changed source files before switching, invoking the debugger, or building an .EXE.

When swapping, QB Plus by default first looks for available extended memory managed by an XMS driver such as HIMEM.SYS. If found, QB Plus allocates an XMS handle of appropriate size and copies QB's conventional memory image, and any loaded files and Quick Library to XMS. If XMS memory is unavailable, QB Plus looks for EMS and if found, attempts to copy to available EMS memory. Otherwise, QB Plus opens a file named QBSWAP.$$$ in the current drive and subdirectory, and swap QB into there if there is sufficient disk space. A command line switch can be used to override swapping to XMS and/or EMS, and instead force swapping to disk only.

You can also change and save the name of the swap file, and the forced disk swap setting in the QB Plus Change Settings menu.

When QuickBASIC is swapped back at the conclusion of the alternate program, the XMS/EMS memory or disk space is released.

Before QuickBASIC is swapped out, QB Plus preserves the current drive and directory, the current video mode, cursor position and size, current screen contents, and interrupt vectors. These are restored on return. Other settings are not preserved, including EMS page mapping, video palette and adapter registers, mouse cursor, and so forth. Changes made to settings that are not preserved and which are not restored by the alternate program when it ends could cause erratic behavior by QuickBASIC Plus or QB. Tests with commonly used commercial software applications have produced no problems in this regard, however.

The Program Switching feature works only with programs that release all of their allocated memory on exit. Terminate and stay resident programs cannot successfully be run with this switcher, and QB Plus will fail when it cannot recover QuickBASIC's memory from the resident program.

Preserving QB Plus/QB Screen Image

If you wish, you can write an external program that appears to be a QB Plus accessory. By specifying the /NOCLS command line argument, or by enabling the corresponding setting in the Settings-Other menu, QB Plus will leave its pop up window with the underlying image of QB on screen whenever the Switch Program (S) feature is invoked. By confining the screen output of your external program to the QB Plus pop up screen and displaying text using the same background color as QB Plus's pop up window, your external program will appear to be a QB Plus accessory. When your external accessory ends and returns control to QB Plus, the original screen is restored. Therefore, your program does not need to preserve it.

Please note that this option to preserve the QB Plus/QB screen image, once selected, will apply to all occasions when the Switch program feature is invoked, regardless of whether the external program called respects the QB Plus window. You will therefore want to disable this feature before switching to such other programs.

Change Settings

The Change Settings of QB Plus lets you customize QB Plus for your current session.

Using the Save Settings option, future sessions may be customized as well, since the stored settings are automatically read from file and incorporated into QB Plus's operation each time QB Plus starts. However, command line options used to start QB Plus take precedence over any default or stored settings. The settings are stored in a file named QBP.CFG located in current directory, or the directory in which QB Plus resides.

Change Settings is accessed from the main QB Plus menu with the C command. From there, select whatever you wish to change—settings for the debugger, macro key player, or others.

Debugger Settings

On the Debugger Settings screen, specify the name of your external debugger in the top box. QB Plus will automatically add an .EXE extension if you do not specify otherwise. You may specify a full path including drive and directory. If you do not give a drive or path, QB Plus assumes the current directory and/or drive. You can edit the default name, CV.EXE. Press Enter when you are finished, or use Tab or Shift+Tab to move to another box.

The next box below is for the command line options you wish to specify to the debugger. Both the debugger name and the options are saved to the configuration file when the Save Settings command is issued, so you do not have to enter these each time QB Plus is loaded.

The lowest box is for the name of the executable program to be debugged. This name is passed by QB Plus to the debugger, along with the command line arguments when you invoke the Debugger command from QB Plus's main menu. The .EXE extension is added to any file name entered without an extension. No path or directory is added by QB Plus if you do not include it as part of the file name.

Unlike the debugger name, the name of the program being debugged is not saved in the QBP.CFG settings file on the premise that it will change from one QB Plus session to another.

Once the data is entered to your satisfaction, move to the <OK> button and press Enter. QB Plus will check to see if the debugger name you specified belongs to a file that actually exists, and will print a brief warning in the top box if the debugger cannot be found.

Once this information has been supplied, you can invoke the debugger with a single keystroke from the main QB Plus menu.

If you have no external debugger, or wish not to use it in this way, you can still make use of this QB Plus accessory. Simply substitute another program name for the debugger name, and this substitute program will be called by QB Plus when you invoke the Debugger command. You can provide substitute command options as well.

Macro Key Customization

The "C" options on the QB Plus Macro Key Settings window provide alternative command key combinations. For the key combination that pops up the QB Plus menu window, press M to toggle between Control+Shift and Left-Shift+Right-Shift. The same alternatives are available for the End Recording command key combination, which are toggled using the E key. Control+Shift is good for single-handed operation, but conflicts with certain QuickBASIC text selection commands. Set these two commands differently if you do not want the QB Plus menu window to pop up when you halt recording.

Playback key options (toggled with P) are Caps Lock, Num Lock, and Scroll Lock. This key command is invoked with a double-tap.

The fourth option on the Change Setting screen controls the delay between playback keystrokes. Each time you press the D key, the delay interval between keystrokes is increased by 1/18th second. After a full second is reached, the interval wraps back to zero. Use a delay increase to slow keystroke playback for trouble shooting, or better timing with QB's execution of recorded commands.

These settings are preserved along with the other QB Plus Settings when you select the Save settings option from the main Change Settings menu.

Other Settings

Various other operating settings may be changed from this window. Use the Tab and Shift+Tab keys to move from box to box.

Pressing Esc at any point, or Enter on the <OK> button will leave the screen and whatever changes you have made intact. If you make these changes on a temporary basis, be sure to restore them before invoking the Save Settings command, because all of these settings are saved to the QBP.CFG and used by QB Plus from then on.

File Swap Name

When XMS or EMS is not available, or is not to be used by QB Plus, the memory image of QB is stored on disk during the Switch Program, Debugger, and Build .EXE external processes. This is the file name QB Plus uses for such disk storage.

This screen lets you change it in the unlikely event of a conflict with another program. This might occur, for example, if you are running on a network, have the current directory set to a public area, and swap your image into that directory at the same time another network user running QB Plus does the same thing using the same swap name you are.

Note that if you are sharing QB Plus on a network, you will also be sharing the QBP.CFG file if it is in a common public directory from which others run QB Plus. To avoid conflicts here, each user should keep QB Plus in a private directory, either on the server, or locally. Since QB Plus first looks for QBP.CFG in the directory in which QBP.EXE resides, your local copy will be used by QB Plus rather than a shared network version. Also, please understand that QB Plus is licensed for one user at a time; additional copies must be purchased if you plan to have more than one programmer using it this way.

There are no provisions for specifying drive and subdirectory, because QB Plus swaps QuickBASIC to the current directory only.

On/Off Toggle Switch Box

With the cursor on the top line within this box, pressing one of the letter keys within the < > brackets will toggle the setting for that item between off and on.

Several of these keys may also be specified as a command line argument, in which case the command line argument will override any setting saved in the .CFG from a prior session.

Toggle Switches are as follows:

Real Time	Continual redisplay of memory contents in the memory viewer when On.
Disk Only	When On, QB Plus swaps the QuickBASIC memory image to disk rather than extended or expanded memory, when an external program is run in Switch Program, Debugger, or Build .EXE.
No Clear Screen	When On, QB Plus leaves its window on screen when an external program is called. Leave this off unless you write a QB Plus add-on that prints only within the QB Plus pop up window.
Port Viewing	When On, the memory viewer will read and display register data from your PC's I/O ports. Leave this off until you can determine that such reading will not be harmful to your PC.
No Direct Extended	When On, prevents QB Plus from entering protected mode to read extended memory directly. This is in case your PC's hardware, BIOS, or software conflict with QB Plus method for doing this and you experience problems in the QB Plus memory viewer.
No EMS Overlays	When On, forces QB Plus to keep its overlays on disk, and not load them into EMS. This frees 80K of EMS, but slows QB Plus's operation somewhat.

Real Time Clock

When On, QB Plus intercepts interrupt 70h, and uses the periodic interrupt feature of a built-in real-time clock to generate profile sampling rates from 32 to 2048 per second. If you set this On and your PC does not have a real-time CMOS clock, the time samples will not be collected at rates above 18 per second.

If the use of the CMOS clock interferes with other software, set this to Off and the clock will not be used. If you save this setting to file, QB Plus will also not intercept interrupt 70h the next time it starts.

Snow suppression

When On, eliminates static encountered on some CGA screens. Leave this Off if you have a CGA screen that does not have this problem.

Ending the Accessories

Because QB Plus runs QuickBASIC as a child process, QB Plus is resident in memory only as long as QuickBASIC is. When you exit QuickBASIC, QB Plus terminates as well releasing its memory back to DOS. In this manner, QB Plus differs from a Terminate and Stay Resident (TSR) utility that is always resident until deinstalled manually.

If you have made changes in settings that you would like to use in the next QB Plus session, be sure to save the current settings before exiting QuickBASIC. The values are stored in the file QBP.CFG in the same directory as QBP.EXE, and loaded by QB Plus at start up.

Before terminating completely, QB Plus checks to see if you have recorded keystrokes that have not been saved, and gives you a last chance to do so.

Appendices

QB Plus Error Condition Codes

It is possible that DOS access by QB Plus may produce error codes.

The following is a comprehensive listing based on DOS Interrupt 21, function 59 (Get Extended Error Information) provided under DOS 3.0 and later. The most likely codes you can expect are 2 and 8 at start up, and codes related to disk conditions when files are written.

DOS ERROR CODE	MEANING
1	Invalid function
2	File not found
3	Path not found
4	No handles available
5	Access denied
6	Invalid handle
7	Memory control blocks destroyed
8	Insufficient memory
9	Invalid memory block address
10	Invalid environment
11	Invalid format
12	Invalid access code
13	Invalid data
14	Reserved
15	Invalid drive
16	Attempt to remove current directory
17	Not the same device
18	No more files
19	Disk write protected
20	Unknown unit
21	Drive not ready
22	Unknown command
23	CRC error
24	Bad request structure length
25	Seek error
26	Unknown media type
27	Sector not found
28	Out of paper
29	Write fault
30	Read fault
31	General failure
32	Sharing violation
33	Lock violation
34	Invalid disk change

DOS ERROR CODE	MEANING
35	FCB unavailable
36	Sharing buffer overflow
37	Reserved
38	Unable to complete file operation
39-49	Reserved
50	Network request not supported
51	Remote computer not listening
52	Duplicate name on network
53	Network name not found
54	Network busy
55	Network device no longer exists
56	Net BIOS command limit exceeded
57	Network adapter error
58	Incorrect network response
59	Unexpected network error
60	Incompatible remote adapter
61	Print queue full
62	Not enough space for print file
63	Print file deleted
64	Network name deleted
65	Access denied
66	Network device type incorrect
67	Network name not found
68	Network name limit exceeded
69	Net BIOS session limit exceeded
70	Temporarily paused
71	Network request not accepted
72	Print or disk redirection is paused
73-79	Reserved
80	File already exists
81	Reserved
82	Cannot make directory entry
83	Fail on Int 24
84	Too many redirections
85	Duplicate redirection
86	Invalid password
87	Invalid parameter
88	Network data default
89	Function not supported by network
90	Required system component not installed

Macro Key Technical Information

1. File structure

The Macro file consists of 36 fixed length records. The length of each record is 83 bytes. Each record begins with a length byte representing the number of valid bytes which follow within the record. 82 is the maximum (two macro bytes plus 80 keystroke bytes). At byte two is the

first macro byte—the ASCII code of the macro key character (A-Z, 0-9). The third byte is the ASCII code of the macro key to which this macro is linked. A code of 127 means there is no link in effect. Up to 40 key codes follow the link byte. They each consist of two bytes: the first is the ASCII code of the character, or a 0 reflecting an extended key code similar to the first character returned by INKEY$ when such a key is pressed. The second byte is the key scan code.

The following QuickBASIC declaration can be used to access the macro file:

```
CONST MaxMacros = 36
TYPE MacroFileT
   Length    AS STRING * 1
   MacroKey  AS STRING * 1
   LinkMacro AS STRING * 1
   Keycodes  AS STRING * 80
END TYPE
DIM Macro(1 TO MaxMacros) AS MacroFileT


ValidMacros$ = "ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890"
```

2. Special codes

The shift, control and alternate keys are identified in a recorded macro with a leading 0 byte, and a second byte as follows:

DECIMAL	HEX	KEY ACTION
255	FF	Alt Down
254	FE	Alt Up
253	FD	Ctrl Down
252	FC	Ctrl Up
251	FB	Left-Shift down
250	FA	Left-Shift up
249	F9	Right-Shift down
248	F8	Right-Shift up

Messages

QB Plus Exit Code [NN], Dos error code [DD].

This message may be accompanied by the message "Program not found" if an error occurred when QB Plus attempted to load QuickBASIC. You may need to change the drive or directory for either QB Plus or

QuickBASIC, or add QuickBASIC's location to the DOS path, or explicitly specify the QuickBASIC path with the /Q: command at QB Plus startup.

If you receive a message "Insufficient Memory", it means there was inadequate conventional memory to start QuickBASIC. You may need to unload other TSR's or drivers that occupy conventional memory.

If only the above message is returned by QB Plus on exit, then a problem condition internal to QB Plus may have occurred. Please note the code number and anything you can of the circumstances occurring during QB Plus's operation, and report the information to Crescent Software.

Cannot continue - Error [00].

QB Plus has encountered a fatal error attempting to read its overlays from file. The reason is listed thereafter as follows:.

Overlay File Not Found.

The QB Plus overlays (.OVR) must be in the same directory as the main QB Plus file, unless the overlays have been combined into a common .EXE file with the main resident code.

I/O error reading overlay file.

The overlay or main .EXE file has become damaged, or the disk itself has become unreadable.

Insufficient heap for overlay.

This reflects a fatal error internal to QB Plus, indicating that the file image of QB Plus may have become damaged. Reload your original copy of QB Plus.

Cannot continue. I/O error reading overlay file.

QB Plus encountered a fatal error attempting to load its overlays into expanded memory. The QB Plus file may be damaged and should be reloaded from the original disks.

Keystrokes recorded in memory may have changed. Save changes (Y/N)?

When QB Plus ends, it provides this prompt as a last chance to preserve recorded, but unsaved keystrokes. Press N to discard them, or Y to be prompted for a filename for saving the keystrokes.

Error detected in configuration file—not loaded.

QB Plus was unable to read its configuration file, QBP.CFG, at startup. Create a new configuration file after QB Plus starts by accessing the Change Settings option, specifying the desired options, then Saving the changes.

Cannot locate QuickBASIC. Program ended.

QB Plus could not locate QB.EXE or QBX.EXE in either the current directory, the DOS path, the directory from which QB Plus is being run, or in a directory specified with the /Q: command line argument. If you are using the /Q: command line switch be sure to include a trailing backslash in the subdirectory name. For example: C:\QB45\.

QB vectors reattached.

Denotes success at reconnecting the swapped-out image of QB/QBC after Switch program.

[XMS][EMS][DISK] swap failed, code [SS]. Operation canceled.

QB Plus encountered a device driver error at the start of the Switch program function when attempting to place a copy of the QuickBASIC image in the specified location. The code [SS] corresponds to error codes returned by the XMS, EMS or Dos disk driver as the case may be.

QB memory shrink failed, code [DD]. Operation canceled.

QB Plus was unable to recover QuickBASIC's memory image area from DOS in preparation for Switching to another program. DOS or its memory control blocks may be damaged. Save your files, reboot, and then reload QB Plus.

DOS exit code [DD]. Press any key to continue.

After a child process called by QB Plus is completed, QB Plus displays any exit code that the child process passed to Dos. An exit code of zero usually means a program ended successfully. Any other code may indicate a problem, or perhaps it represents information being returned to the calling program. Consult the child program's documentation for possible meanings of its exit code.

Error code [DD] restoring subdirectory [D:path].

QB Plus is unable to reset the current directory to the same drive and directory that was the current directory at the time Switch program function was invoked. There may be a problem with the disk drive, or the disk may have been removed.

[XMS][EMS][DISK] swap failed [SS]. QB is unrecoverable.

Following the Switch Program process, QB Plus was unable to retrieve QuickBASIC's stored memory image. This message will also indicate the source driver where the problem was encountered as XMS, EMS, or disk, and the error code [SS] corresponding to the particular driver error.

[ProgramName] not found. Operation canceled.

During the Switch program process, the specified program could not be found in either the current directory, or along the DOS path. An .EXE extension is assumed and added to any filename you enter without an EXE extension. .COM files must have the .COM extension explicitly included in the entry you provide. Batch (.BAT) files cannot be run from the Switch Program menu in QB Plus.

[Debugger FileName] not found. Operation canceled.

QB Plus could not find your debugger in either the current directory or a directory in the Dos path list. If the .COM or .EXE extension is not specified, QB Plus adds an .EXE extension to the file name you enter. A complete drive and path may be entered along with the debugger name.

Error [DD] attempting to save data.

QB Plus has encountered an I/O error when saving Compile/Link options to disk.

Code [DD]. Cannot read file.

QB Plus encountered an I/O error attempting to read a macro key file.

Code [DD]. Cannot write file.

QB Plus encountered an I/O error attempting to write a macro key file.

Printer error. Retry(Y/N)?

An error has been returned by DOS when printing to the printer. Correct the problem (out of paper, printer off, printer off line), then respond with Y to resume printing.

Sample Buffer Overflow.

During profiling, the profiler's resident memory buffer capacity of 281 routines was exceeded and the excess routines have not been sampled. This is just a warning error.

QBPBUILD.EXE not found.

During the Build .EXE process, QB Plus was unable to locate the external companion program QBPBUILD.EXE in either the current directory, the DOS path, or the directory from which QB Plus was run.

Run time error [RRR] at [SSSS:OOOO].

An internal error fatal to QB Plus has occurred at the segment and address indicated.

Problem Conditions

QB Plus hangs on startup

If the problem occurs before QB Plus displays its startup message, you may have a different version of QB Plus in the current directory than the version you actually started in a different directory. Remove one version or run QB Plus from the current directory.

If the problem occurs after the startup message, there may be a problem with either an EMS or XMS driver. Try running QB Plus without either or both.

QB Plus does not respond to its pop up keys

QB.EXE or QBX.EXE may not be in the editing mode waiting for a keystroke entry. QB Plus will not pop up while a program is running, or while a QB/QBX menu is active. This includes the top line menu bar which is easily activated by merely pressing Alt.

Other possibilities are that QB Plus did not recognize the version of QuickBASIC that it loaded, the computer's BIOS is not 100% IBM compatible, or input has been redirected by another resident program or the operating system. It is also possible that the keyboard is stuck in a shifted state (as if the Alt, Shift or Ctrl were being invisibly held down). Try tapping all Alt, Shift, and Ctrl keys, try the Ctrl+Break key combination, remove any TSR's, try a different version of DOS, or ensure that QB Plus supports your version of QuickBASIC.

QB Plus crashes, hangs, or reboots when the Memory Viewer option is invoked

There is a protected mode memory access conflict between QB Plus and your PC's BIOS, a loaded memory manager, or operating system. Try starting QB Plus with the /NX option.

QB Plus reports "Cannot find [filename].EXE" for a .COM file

In Switch Program, an EXE extension is added to any program name entered without an extension. For .COM files, you must specify the full file name including the .COM extension.

Profiling does not work

If samples are not collected during profiling, first be sure profiling was enabled—that is, you set profiling to Enabled, and then you exited the profile menu using the Enter and not the Esc key.

If you were using a sampling rate greater than 18, your PC must have an AT-compatible real time clock that QB Plus can set to generate a periodic interrupt. Otherwise, the QB Plus sample collection routine will not be invoked. Some operating systems, such as OS/2, Windows,

and DesqView may prevent this periodic interrupt from reaching QB Plus, and you will be able to sample only at 18 samples per second rate under those operating systems.

Profiling rate cannot be changed in the profiler menu

If the sampling rate is 18, and pressing the R key does not change it, then QB Plus is set up to ignore any AT-compatible real time CMOS clock in your PC. This occurred either because QB Plus could not identify an appropriate clock; or a real-time clock was found but the /NORTC switch was contained in the QB Plus configuration file or was given as a QB Plus option.

Profiling results are inconsistent from one run to the next

The profiler bases its analysis on sample data, not exact measurements. Sampling rates and sample counts, along with the relative frequency that a routine was sampled, all affect the precision of the results. Some minor variation should not be an obstacle to optimizing your program, as you will need to improve speed by large amounts to have noticeable affect on the overall results.

To obtain the most precise measurements, use the highest sampling rate, run the program long enough to get a representative sample collection, and do several sampling runs combining the results of the runs by not discarding samples between runs. Also be aware of possible harmonics and stroboscopic effects; and be certain you have not inadvertently enabled or disabled debugging features that change the program execution rate between comparison runs. See the Profiling section of the manual for details.

Main module duration increases the longer the profile sample is taken

The main module, while called only once at the start of the program, may be returned to many times during execution of the program. Thus, in a program where there are continued calls to subroutines from the main module, the main module duration will gradually increase as the program runs.

QB Plus does not correctly report installed extended memory with the F3 key

QB Plus uses the Int 15 BIOS call, as well as an inspection of the CMOS configuration area in AT-compatible PC's to report memory. In some PC's with additional extended memory on a peripheral card (such as certain PS/2 models), the data on the additional memory may be stored at a location QB Plus does not access. You should still be able to view the additional memory locations in the memory viewer.

QB Plus uses the wrong name for the loaded QuickBASIC program

Significant changes to a loaded program, such as changing its name, changing main module designations, loading and unloading various modules, and extensive editing can sometimes confuse QB Plus. In such a case, save your program and its modules, then reload them fresh from disk, beginning with the main module. This will usually correct the situation. If not, end QB/QB Plus, restart it, and finally reload your program.

QB Plus seems to remove certain QB/QBX command line parameters

If you use the /CMD switch for QuickBASIC when starting QB Plus, command arguments which follow /CMD and normally belong to QB Plus will be accepted by QB Plus as its own, and removed from the command line before the line is passed on to QB/QBX. As a workaround, try using the "-" switch character, or place a space between the switch character and the argument, for arguments following /CMD. Alternatively, set the /CMD switches in the QB/QBX Run-Command menu.

32 Seventy Acres ■ West Redding, Connecticut ■ 06896 ■ 203-438-5300